NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

FAULT-TOLERANT SEQUENCER USING FPGA-BASED LOGIC DESIGNS FOR SPACE APPLICATIONS

by

Jason J. Brandt

December 2013

Thesis Co-Advisors: Herschel H. Loomis, Jr.
James H. Newman

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 2013	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE FAULT-TOLERANT SEQUENCER USING FPGA-BASED LOGIC DESIGNS FOR SPACE APPLICATIONS	5. FUNDING NUMBERS
6. AUTHOR(S) Jason J. Brandt	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. government. IRB protocol number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	12b. DISTRIBUTION CODE A

13. ABSTRACT (maximum 200 words)

The design of a device that controls the sequence and timing of deployment of CubeSats on the Naval Postgraduate School's CubeSat Launcher (NPSCuL) is detailed in this thesis. This design is intended to be implemented on a field-programmable gate array (FPGA) installed into the NPSCuL. This configuration allows flexibility in reprogramming the launch sequence and adding additional functionality in future designs.

Operating an FPGA on orbit presents unique challenges due to the radiation environment. Radiation from space cannot be shielded efficiently, so devices must be tolerant of the expected effects. The most common effect, the single-event upset can have detrimental effects on operating electronics, causing undesired changes to data.

To combat this problem, fault tolerant techniques, such as triple-modular redundancy (TMR) are explored. In these methods, multiple redundant copies of the design are operated simultaneously, and the outputs are voted on by special circuits to eliminate errors. Comparisons between manual and software generated TMR methods are tested, and the design is implemented on test hardware for further verification. Finally, future research and testing is discussed to continue to ready the design for employment of the sequencer on an actual space mission.

14. SUBJECT TERMS Single-Event Effect (SEE), Single-Event Upset (SEU), Multiple-Bit Upset (MBU), Field Programmable Gate Array (FPGA), Fault Tolerance, Triple Modular Redundancy (TMR), Quadruple Force Decide Redundancy (QFDR), Quadded Logic, CubeSat, Satellite, Actel, Microsemi, ProASIC3, Xilinx, Virtex, Synplify	15. NUMBER OF PAGES 159
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

FAULT-TOLERANT SEQUENCER USING FPGA-BASED LOGIC DESIGNS FOR SPACE APPLICATIONS

Jason J. Brandt
Lieutenant Commander, United States Navy
B.S., University of Colorado, 2002

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2013

Author: Jason J. Brandt

Approved by: Herschel H. Loomis. Jr.
Thesis Co-Advisor

James H. Newman
Thesis Co-Advisor

Clark Robertson
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The design of a device that controls the sequence and timing of deployment of CubeSats on the Naval Postgraduate School's CubeSat Launcher (NPSCuL) is detailed in this thesis. This design is intended to be implemented on a field-programmable gate array (FPGA) installed into the NPSCuL. This configuration allows flexibility in reprogramming the launch sequence and adding additional functionality in future designs.

Operating an FPGA on orbit presents unique challenges due to the radiation environment. Radiation from space cannot be shielded efficiently, so devices must be tolerant of the expected effects. The most common effect, the single-event upset can have detrimental effects on operating electronics, causing undesired changes to data.

To combat this problem, fault tolerant techniques, such as triple-modular redundancy (TMR) are explored. In these methods, multiple redundant copies of the design are operated simultaneously, and the outputs are voted on by special circuits to eliminate errors. Comparisons between manual and software generated TMR methods are tested, and the design is implemented on test hardware for further verification. Finally, future research and testing is discussed to continue to ready the design for employment of the sequencer on an actual space mission.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

3D	three dimensional
AMU	atomic mass unit
ASIC	application-specific integrated circuit
BGA	ball grid array
BTMR	block TMR
CAD	computer-aided drafting
CAM	computer-aided manufacturing
CCC	clock conditioning circuit
CFTP	Configurable Fault-Tolerant Processor
CLB	configurable logic block
CMOS	complimentary metal-oxide semiconductor
COTS	commercial off-the-shelf
CPLD	complex programmable logic device
CPU	central processing unit
CS	compton scattering
DC	direct current
DIP	dual-inline package
DSP	digital signal processing
DTMR	distributed TMR
EAR	Export Administration Regulation
EEPROM	electrically erasable PROM
EHP	electron-hole-pair
EMI	electromagnetic interference
FF	flip-flop
FPD	field programmable device
FPGA	Field Programmable Gate Array
FROM	flash ROM
FSM	finite-state machine
GHz	gigahertz
GND	ground

GTMR	global TMR
HDL	hardware description language
I/O	input/output
IC	integrated circuit
IEEE	Institute of Electrical and Electronics Engineers
IP	intellectual property
ISE	Integrated Software Environment
ISim	ISE Simulator
ITAR	International Traffic in Arms Regulations
JTAG	Joint Test Action Group
LED	light-emitting diode
LFSR	linear feedback shift register
LSB	least significant bit
LT	lieutenant
LTMR	local TMR
LUT	lookup table
LVDS	low-voltage differential signaling
MBU	multiple bit upset
MOSFET	metal oxide semiconductor field effect transistor
MSB	most significant bit
NASA	National Aeronautics and Space Administration
NPS	Naval Postgraduate School
NPSCuL	NPS CubeSat Launcher
OTA	over-the-air
P&R	place and route
PA3TB	ProASIC3 Test Board
PAL	programmable array logic
PCB	printed circuit board
PLA	programmable logic array
PLD	programmable logic device
PLL	phase-locked loop
PP	pair production

PPE	photo-electric effect
P-POD	Poly-Picosatellite Orbital Deployer
PROM	programmable read-only memory
PSRR	power supply rejection ratio
QFDR	quadruple force decide redundancy
Rad-Hard	radiation hardened
RAM	random-access memory
RHBD	radiation hardening by design
RTL	register-transfer level
RTOS	real-time operating system
SAD	splitter auxiliary device
SADv3	SAD Version 3—Flight Prototype Board
SBU	single bit upset
SDK	software development kit
SDRAM	synchronous dynamic random-access memory
SEB	single-event burnout
SECDED	single error correct, double error detect
SEE	single-event effect
SEFI	single-event function interrupt
SEL	single-event latchup
SET	single-event transient
SEU	single event upset
SMT	surface-mount technology
SoC	system on chip
SOI	silicon on insulator
SOS	silicon on sapphire
SRAM	static random-access memory
TID	total ionizing dose
TIR	triplicated interwoven redundancy
TMR	triple modular redundancy
TSOP	thin small-outline package
TTL	transistor-transistor logic

USB	universal serial bus
V_{CC}	positive voltage source
V_{DC}	volt(s) direct current
VHDL	VHSIC hardware description language
VHSIC	very-high-speed integrated circuits
VLSI	very-large-scale integration
VQFP	very thin quad flat package
XST	Xilinx Synthesis Tool

EXECUTIVE SUMMARY

The purpose of this research was to detail the design of a control system to deploy CubeSats from the Naval Postgraduate School CubeSat Launcher (NPSCuL). This design was developed as a minimal design with no inherent redundancy and then expanded to redundant versions with various types of fault tolerance. Comparisons in the operation between these versions as well as the required hardware resources to handle the additional components were performed.

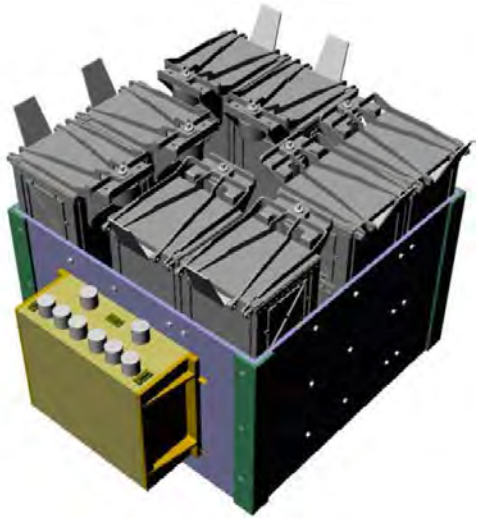


Figure 1 NPSCuL 3D model, showing eight P-PODs and the attached electronics enclosure.

The NPSCuL is a cube-shaped structure, about 20 inches per side. It is designed to be mounted on the aft end of an Atlas V Centaur launch vehicle. It holds eight poly-picosatellite orbital deployers (P-PODS), and each P-POD can hold up to three small cube satellites, each about four inches per side. A three-dimensional (3D) model of the NPSCuL can be seen in Figure 1. When the host spacecraft reaches the desired orbit, the P-POD doors are opened, and the picosatellites are deployed by a spring deployment mechanism. The NPSCuL either controls the deployment sequence via an onboard control system, known as the sequencer, or the deployment is controlled by the launch vehicle's flight computer. The sequencer logic is programmed into a field programmable

gate array (FPGA). This design choice allows a reduced cost over a custom application specific integrated circuit (ASIC) and allows for design flexibility as capabilities are changed or added in the future.

The environment of space produces a complex and varied radiation environment. Objects in space are bombarded by high energy gamma rays, streams of protons, neutrons, and electrons. Impacts from alpha particles and heavier elements are continuously occurring. Any spacecraft must be able to handle these conditions in order to operate reliably. While many materials are largely unaffected by these conditions, electronics can experience significant detrimental effects. The energy generated by each impact in the silicon of the transistors are known as single-event effects (SEEs). These effects can cause a transient pulse, called a single-event transient (SET), or even cause permanent damage to the device in the form of a burnout or gate rupture. The SEE of most concern here is the single-event upset (SEU), where an SEE causes a bit flip in a register or memory location from a one to a zero or vice versa. An SEU in an operating device such as a spacecraft control system can have disastrous results and lead to mission failure or even loss of the spacecraft.

To protect against SEUs and other SEEs, the electronics manufacturing industry has developed two primary methods. One involves designing their devices with different materials and methods that are more resistant to radiation effects. This includes manufacturing techniques such as silicon on insulator or silicon on sapphire, where extra insulation material is incorporated into the semiconductor, increasing the energy threshold required to cause an SEU. These design changes are effective but can be very costly. The second protection method adds redundancy or error detection and correction methods to the designs themselves. Using multiple devices or multiple segments of logic on the same device ensures that a single radiation event does not affect the entire system, and the remaining unaffected units can continue to operate correctly. This redundancy can take the form of quadded logic, quadruple force decide redundancy (QFDR), triple interwoven redundancy (TIR), or the triple modular redundancy (TMR). Other error correction and detection schemes such as error correcting codes (ECC) and reduced

precision redundancy (RPR) are also used in some devices. Due to their reconfigurable nature, FPGAs are susceptible to SEEs in their normal operation as well as their configuration memory.

The NPSCuL will be operating in this environment once launched, and its electronics must be able to handle these conditions. Prior flights of the NPSCuL have used the launch vehicle to control deployments. Future flights are expected to use the onboard sequencer. The sequencer design presented here should meet the mission requirements, provide sufficient fault tolerance to operate reliably on orbit, and provide flexibility for future operations.

To begin the design of the sequencer, the operational, mechanical, and electrical requirements for the system were obtained from the various design documents. These include operating voltages, time delay requirements, interfaces, and mechanical clearances. From these requirements, a functional flowchart was generated, and a finite state machine was developed. The design was created with separate modules, which allows more flexibility for future design changes and improvements, as seen in Figure 2. Additional supporting hardware was also designed, specifically a radiation hardened launch command voter device, which serves as a final launch command decision to eliminate faults caused by an SEU in the FPGA outputs.

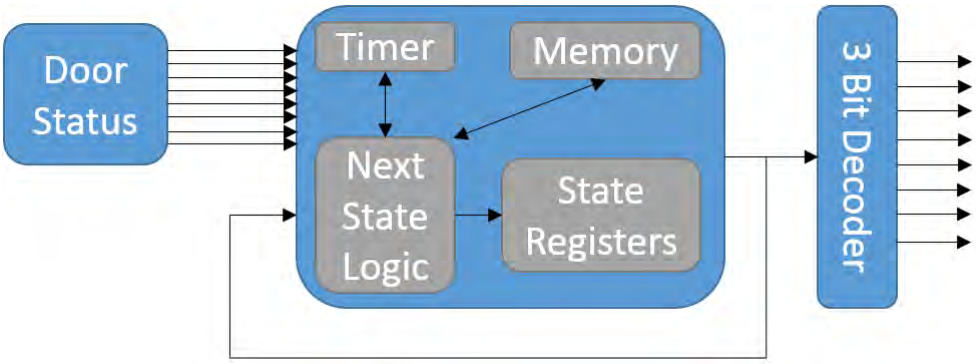


Figure 2 Block diagram for the single-sequencer design.

To properly test this sequencer design, it first had to be implemented in software. After a brief review of appropriate languages and development environments, the design

was created in Verilog, a hardware description language. With its flexible and powerful simulation tools, the Xilinx Integrated Software Environment (ISE[1]) was chosen as the development environment. Selecting the appropriate testing methods and techniques was critical to verify proper operation of the fault tolerant features of the sequencer. A list of potential errors was developed that encompassed the most likely SEE errors that would be seen on orbit. These included errors such as bit-flips, stuck-at-one or zero errors, and floating inputs and outputs. A small fault module was created and used to insert these various fault types into the design. An analysis of the most significant error locations was performed, and the fault modules were inserted into those locations.

Fault performance testing was performed on the single sequencer design first with no included redundancy. This establishes a baseline of performance and demonstrates the types of errors that are expected in a non-fault tolerant design. Two separate fault-tolerant design versions were then created. The first using a distributed TMR method done manually, as shown in Figure 3. The second used the automatic TMR features of a software program by Synopsys called Synplify[2]. These two versions were tested against the same fault sequence as the initial design version. In every case, the manual TMR version was able to correct the faults before the output was affected, proving the effectiveness of the TMR concept. The software designed version was effective with some errors, but not all. The software-assisted TMR design was considerably quicker to create than the manual TMR design, taking only a few seconds to complete. The final result was lacking in fault tolerance compared to the manual design, however, and consumed nearly the same degree of FPGA resources. For simple designs, manual TMR has an advantage. As complexity increases, the designer may need make a decision between the man-hour cost of development and the level of fault tolerance provided.

[1] ISE® is a registered trademark of Xilinx

[2] Synplify® is a registered trademark of Synopsys, Inc.

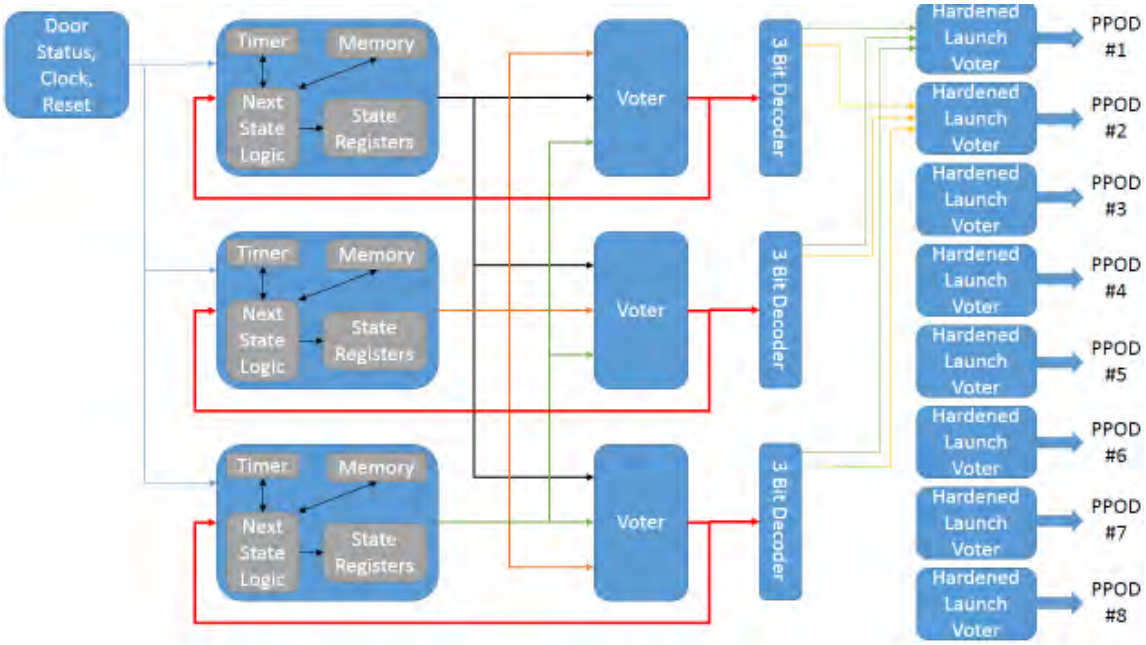


Figure 3 Block diagram for manual distributed TMR sequencer design, showing triplicated logic and voter locations.

Following this development, the software design was translated onto physical FPGA hardware. An FPGA hardware prototyping and development board was selected, the Genesys[3] board by Digilent. This board provided a good mix of light emitting diodes (LEDs) for indications and on-board switches and pushbuttons to properly provide input to the system. A comparison of the required hardware resources demonstrated that the TMR version does require significantly more resources than the single version. While this is not a major factor in this example, it is a concern for anyone designing a more complex or resource intensive project.

Testing demonstrated that the manual TMR version of this design is resistant to all the error conditions applied. The software-assisted TMR design was good but still did not protect against some of the SEE-induced faults that the manual TMR did, despite the similar level of resources consumed.

Further research is needed to determine the effect of an actual radiation field on the sequencer design, to compare other radiation tolerant logic methods, such as

[3] Genesys® is a trademark of Digilent, Inc.

intelligent place-and-route, and to explore future capabilities and uses of the FPGA design on the NPSCuL. The goals of this project were met, with a successful design of a SEE-resistant sequencer suitable for operation on the NPSCuL.

ACKNOWLEDGMENTS

I would like to first thank my wife, Kate for her support and encouragement. Despite the long hours and countless weekends of work, she has always stood by me and helped me to achieve this life-long goal. Second I would like to thank my daughter, Isabelle, for always cheering me up after a long day, and providing artwork and gifts for my workspace.

Special thanks are also offered to the following people:

To Dr. Hersch Loomis for his guidance, advice and support, and Dr. Jim Newman for his input and perspective.

To my classmates in Space Systems Engineering, Adam Sears, Sarah Bergman, Henry "Longley" Thomason, and Greg Contreras for providing entertainment, conversation, and lunchtime chatter within the "circle of trust."

And finally, to my daughter's pet fish, Spinester, for cleaning the fish tank walls of algae, lighting my tank-cleaning workload.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PURPOSE

The purpose of this research is to compare manual and automated means of creating reliable sequential logic designs using fault-tolerant methods. While manual methods are proven, they can be very time consuming to develop and test. Automated means via new commercial software packages are available and have the potential to generate significant cost and time savings. The eventual goal of this research is to produce logic designs that can operate reliably in high radiation space environments using lower cost military or commercial aeronautical grade hardware rather than expensive radiation-hardened (rad-hard) devices.

In order to achieve high reliability against the types of radiation-induced faults that are encountered in an operating environment, several different methods are typically employed. A device that suffers permanent damage after a short time in a radiation environment is obviously unsuitable. On the other extreme, radiation hardening of the hardware itself through design or shielding is cost and weight prohibitive in this application. For any operation in space, some level of radiation resistance or tolerance is required. Fault tolerance via logic design methods, such as triple modular redundancy (TMR) or quadded logic, are well-known methods but labor intensive. Even for simple sequential machines, the overall complexity increases exponentially. As a result, these methods are not frequently employed for anything but the simplest of logic designs. New software packages, such as Precision[1] Synthesis RTL Plus by Mentor Graphics, TMRTool[2] by Xilinx and Synplify[3] Premiere by Synopsys, enable a designer to apply these high-reliability techniques with software tools. This makes much more complex designs, up to full reduced instruction set-type CPUs or system-on-a-chip (SoCs) devices, a possibility.

---

[1] Precision® is a registered trademark of Mentor Graphics Corporation

[2] TMRTool® is a registered trademark of Xilinx, Inc.

[3] Synplify® is a registered trademark of Synopsys, Inc

While this research can be applied to any general sequential or combinational logic machine, the specific application is for a launch sequencer for the Naval Postgraduate School (NPS) CubeSat Launcher known as NPSCuL [1]. This platform is an excellent fit for this type of design due to the complexity of the design, the requirements for operation, and the association with NPS. A primary driver of most CubeSat programs is lower costs and the use of commercial field programmable gate arrays (FPGAs) to achieve levels of reliability comparable to rad-hard devices would be highly desirable. The same concepts apply to the launch platform for the CubeSats, where a launch sequencer must be highly reliable, reconfigurable, and low cost. Current commercial products that meet the design goals are far too costly to be realistically considered.

NPSCuL offers an excellent platform to begin to expand these concepts to physical hardware in a real operating environment. The sequencer has a limited number of inputs and outputs and must simply execute a timed launch sequence for the installed CubeSats when initially powered on. The actual sequence and timing for each launch is programmable on the ground prior to launch. No outside commands into the system are provided; the sequence starts automatically when the system is powered. The outputs are the individual P-POD launch commands and status reports for each launch. This design is developed to easily integrate with the existing hardware and requirements developed in previous work on this launcher. In particular, the design created in this thesis must work with the FPGA and printed circuit board (PCB) designs that have already been developed for the NPSCuL [2].

B. PREVIOUS WORK

The NPSCuL sequencer and supporting hardware, software and components have been developed by a number of students over the course of the program's history. Each person's work has been useful to develop the concepts and technology to allow the program to reach its current status. The key portions of this work relevant to this thesis are listed here.

1. NPS Configurable Fault-Tolerant Processor Project

The Configurable Fault-Tolerant Processor (CFTP) project [3], [4], [5], [6] was a cornerstone of the research by NPS on radiation effects on FPGAs. This project was worked on by multiple students and staff over several years of research. The project mainly focused on considerably more complex logic designs, up to and including complete general processor units. The individuals working on these projects developed an excellent knowledge base of information for implementing fault-tolerant designs in hardware and were able to perform some testing in an actual radiation field. Since much of the CFTP data was obtained using older generation FPGA hardware, the results with newer generation hardware can be expected to change. In addition, the reduced complexity of the sequencer design and the robustness of NPSCuL interfaces should lead to fewer expected faults.

2. Parobek's Hardware

Lieutenant (LT) Luke Parobek developed a test board known as the "ProASIC3 Test Board" [2] for testing of a proposed sequencer implementation. This small PCB incorporated a commercial FPGA from Actel and serves as a test bed and a preliminary prototype for future development of a flight-ready sequencer. In addition, the details for the interface of this sequencer with existing NPSCuL electronics was developed to allow future integration. This hardware is robust and flexible and is essential for initial verification and testing of various logic designs.

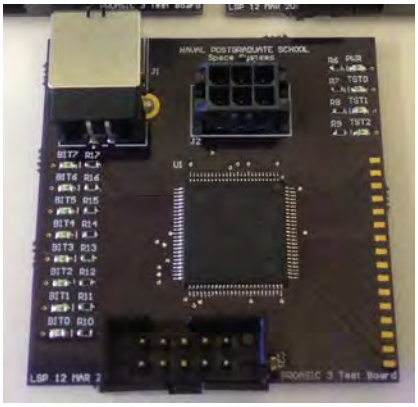


Figure 1. ProASIC3 test board (from [2])

3. Majewicz's TMR Research

LT Peter Majewicz performed significant research into the implementation of TMR logic designs [7]. While his work involved significantly more complex designs than this thesis, the overall concepts are still very relevant for this work. In particular, he was able to perform actual radiation testing at the University of California-Davis' cyclotron. This data can be used to estimate performance of future logic designs that use his methods.

C. NPSCUL AND SEQUENCER HISTORY

1. NPS Cube Sat Launcher

A CubeSat is a class of satellite known as a nanosatellite, with a mass between one and 10 kilograms. These small satellites are currently used for space research by academic and commercial groups [8]. The smallest of these devices is a "1U" size, which is defined as a cube 10 cm on each side [9]. A 2U or 3U size shares the same base as a 1U size but either doubles or triples, respectively, in height.

Figure 2. Typical 1U CubeSat (from [8])

NPSCuL is a CubeSat launcher, developed by NPS and funded by the National Reconnaissance Office (NRO), as shown in Figure 3. This relatively small package is designed to fit within a small volume, mounted as a secondary payload. The launcher consists of an aluminum framework and base and a minimal amount of supporting electronics. The framework supports eight poly-picosatellite orbital deployers (P-PODS), each of which is capable of deploying up to three 1U CubeSats or combinations of 1U, 2U or a single 3U CubeSat. The base is compatible with an evolved expendable launch vehicle (EELV) secondary payload adapter (ESPA), which allows mounting to a variety of primary payloads for delivery to orbit. The electrical interface with the host is limited to some number of power and data signals. In one possible configuration, the launch vehicle provides all the required electrical power and door switch data lines. In this case, the host controls the launch sequence and timing for each P-POD and each P-POD's door status is directly available back to the launch vehicle. In another possible configuration, only primary and secondary power and a single data line are provided by the host. In this configuration, the sequencer is necessary to launch the P-PODs and monitor the status of each P-POD door position [1].

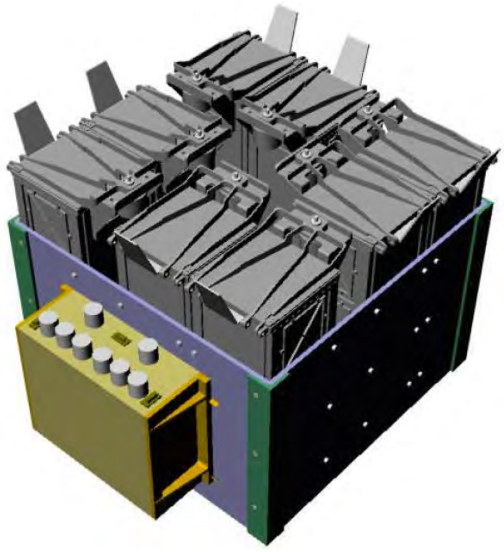


Figure 3. NPSCuL 3D model, showing P-PODs mounted inside frame and attached sequencer and control electronics enclosure (from [10])

2. NPSCuL Sequencer

The sequencer has several unique requirements that led to the selection of an FPGA based control system. In particular, the design must be programmable and reconfigurable for future missions and requirements. The P-POD launch order and timing between each launch can be changed at any time prior to launch while physical access to the NPSCuL is still possible. The sequencer must also fit within the existing physical volume of the NPSCuL electronics structure, as depicted by the yellow box in Figure 3, as well as interface electrically with the current hardware without major modifications. The operation of the launcher when under the control of the sequencer must be identical to operation under the control of the primary spacecraft. The sequencer will be exposed to the same radiation environment as the primary spacecraft, and must operate without radiation induced errors under these conditions. The ability to reconfigure the control system via an FPGA design is the key factor in this system due to the flexibility that it provides to the engineers to meet changing requirements and implement new features and capabilities without necessarily requiring an extensive redesign or expensive testing of the hardware.

D. THESIS ORGANIZATION

This thesis is divided into six chapters. The first chapter presented the purpose of this work, previous related work completed here at NPS, and the history of the NPSCuL and the associated sequencer. In the second chapter, background information on the radiation environment encountered by spacecraft on orbit is discussed. A basic overview of the effects of radiation on semiconductor electronics is also explained, as well as the efforts by engineers to mitigate those effects through new hardware designs, and finishes with a brief review of the most common types of logical fault tolerance. The third chapter begins with an explanation of the NPSCuL sequencer's operational requirements and design decisions and then proceeds to detail the process used to generate a design from those requirements. The concept, methods, and results of testing for the sequencer design are contained in Chapter IV. The basic sequencer's functionality is verified and compared to that of the chosen fault tolerant methods in this chapter. Translating these

designs into a physical form on hardware devices is detailed in Chapter V, and a comparison of the required resources for each design is made. Final conclusions and recommendations for future work and continued development can be found in Chapter VI. The final pages contain the appendices, which contain the source code and relevant schematics generated during the course of this research.

THIS PAGE INTENTIONALLY LEFT BLANK

II. FAULT TOLERANCE

The nature of space flight puts very strict requirements on any spacecraft and its associated components. After these systems are built and tested, they must survive harsh launch conditions and the transition to vacuum and space. Then they must power on independently, establish control of themselves and establish communications with the ground. Once on-orbit, the hardware systems must operate reliably for their mission life, usually with no possibility of repairs or maintenance. Few other highly complex systems must meet these requirements. The occurrence of electronic faults and how those faults are handled is, therefore, a very important area of discussion within the space community.

Electronic devices operating in space are subject to an extremely harsh operating environment. Space qualified components typically operate in a vacuum, experience extreme temperature variations while operating, undergo heavy vibration and shock during launch, and function in widely varying radiation conditions. Most consumer electronics cannot survive these conditions. Even more expensive industrial and aerospace-specific parts may not be able to function in these conditions. In response to the challenges, electronics manufactures have developed very specific parts. While these parts are often engineering marvels, they come with a very high price tag. In particular, the radiation hardened devices command a premium price, orders of magnitude greater than commercial-off-the-shelf (COTS) aerospace devices. Temperature, atmospherics, and vibration are a concern for almost all industrial electronics; however, the impact and effects from a radiation environment is fairly unique to spacecraft and submarines and serves as the focus of this thesis.

A. RADIATION ENVIRONMENT AND EFFECTS

1. Radiation Environment on Orbit

The sun produces radiation at all energy levels. A fairly constant source is emitted by the solar wind. The sun also produces random solar flares and coronal mass ejections (CMEs), which can produce shorter term but much higher intensity bursts of energy. The complex magnetic fields of the Earth can be seen along with some typical

satellite orbits in Figure 4. These fields trap, concentrate and accelerate radiation particles and are still not well understood by NASA. Some radiation particles even arrive from other stars in deep space. These particles can be categorized into several types and energy levels.

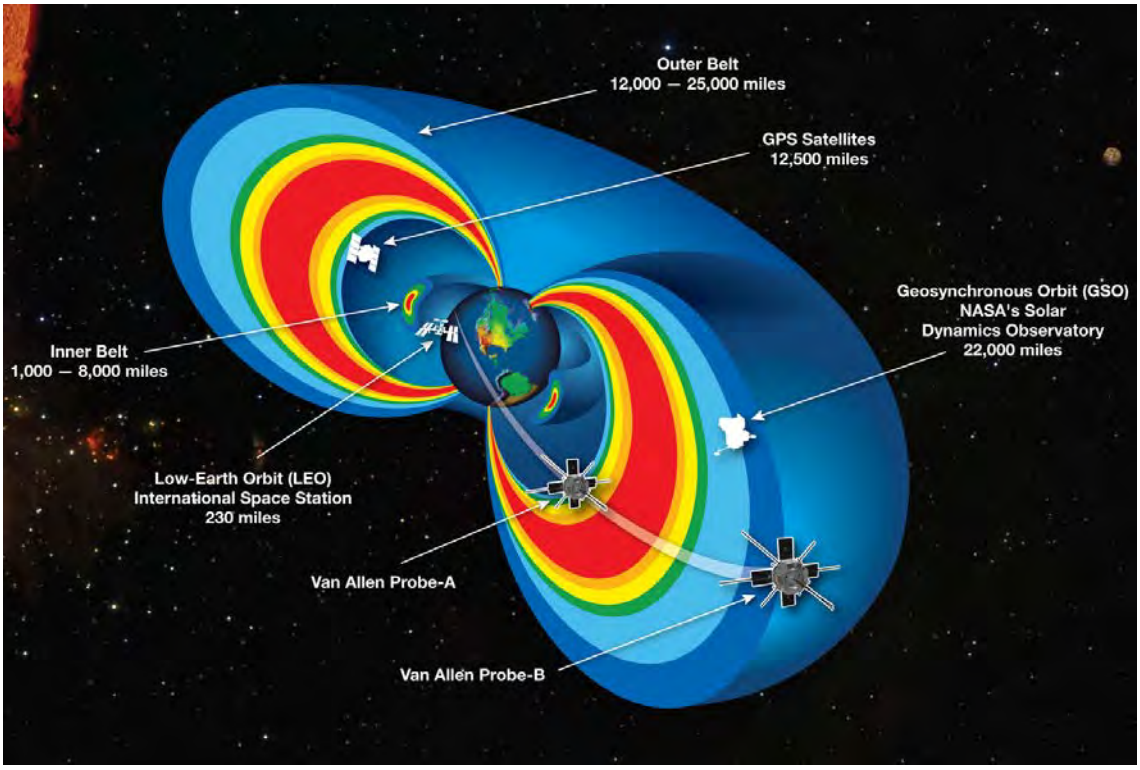


Figure 4. The Van Allen radiation belts around the Earth are being studied and characterized by NASA (from [11])

High energy electrons are usually generated by the magnetic fields of the Earth's Van Allen belts. These belts of magnetic force accelerate lower energy electrons to much higher energy states. Electrons are also found in the solar wind, solar flares, and CME material. Similar to electrons, protons are produced by the same sources. Due to their higher mass, they can produce more damaging effects for the same particle velocity.

Heavy elements, primarily produced by solar flare events, can do a great deal of damage to electronics and generate second and third order events due to their high velocity, high electrostatic charge, and their high mass. The mass of individual particles,

the energy, and the concentration can vary wildly in a short time. Radiation from sources outside our solar system, known as cosmic radiation, is also a significant source of heavy ions. This name can be a misnomer, as it does cover nearly any radiation heavier than an alpha particle and not just heavy metals.

Gamma rays, neutrons, and other particles of varying energy and density also exist in the space environment and must be considered. Even the structural materials of the spacecraft can become irradiated over time and emit radiation. A comparison of the relative strength of the incident particles and energy can be seen in Table 1.

Table 1. Comparison of mass and charge between radiation particles encountered on orbit.

Particle Type	**Mass (in amu)**	**Charge (in e)**
Electrons	5.48×10^{-4}	-1
Protons	1	1
Neutrons	1	0
Alpha	4	+2
Heavy Element	Typically >4	Typically >+2
Gamma Ray	0	0

While operating on earth, the majority of this radiation is blocked by the Earth's magnetic field and atmosphere. Once any spacecraft is launched, however, it loses the shielding of the atmosphere and must deal with these new surroundings. The structure of a spacecraft and the protective covers of the equipment boxes provide shielding from the majority of the low-energy radiation electronic components encounter. Higher energy gamma rays and particles still penetrate these materials, and significant levels of shielding are impractical for spaceflight due to the mass constraints for launch. Even with modern field-programmable gate array (FPGA) and application specific integrated circuit (ASIC) devices, the very compounds that form their packing contain trace amounts of radioactive elements and can emit alpha particles as they decay. Since the radiation environment is impossible to avoid, the sensitive electronic components must be designed to handle these conditions.

2. Radiation Interactions with Electronic Materials

To determine the effects radiation has on a specific electronic component, it is necessary to first define some terms and concepts. The term “fluence” is defined as the number of particles passing through a given area and is expressed in particles/cm2. When fluence is measured over a given time, the term is given as flux and is expressed in particles/cm2-sec. To determine the total radiation that a material has absorbed, the unit of rad (radiation absorbed dose) is used, and it is defined as 100 ergs per gram of energy absorbed by the given material. The amount of energy that is actually absorbed is dependent on the type of radiation as well as the material in question, so it is frequently annotated as such. For silicon, it is noted as rad(Si)

The effects of radiation on any complex electronic devices can be divided into two potential categories, regardless of radiation source. One category is ionizing effects, which primarily occur from photon radiation, and the second is displacement effects, which usually occur from charged particle or neutron radiation. With the uncertain nature of nuclear and radiological interactions, there is no way to guarantee or predict exactly when or how an interaction will occur. It depends heavily on the material involved, the type and intensity of the radiation, and the energy levels. Effects are, therefore, generally described with a probability of occurring in a given situation.

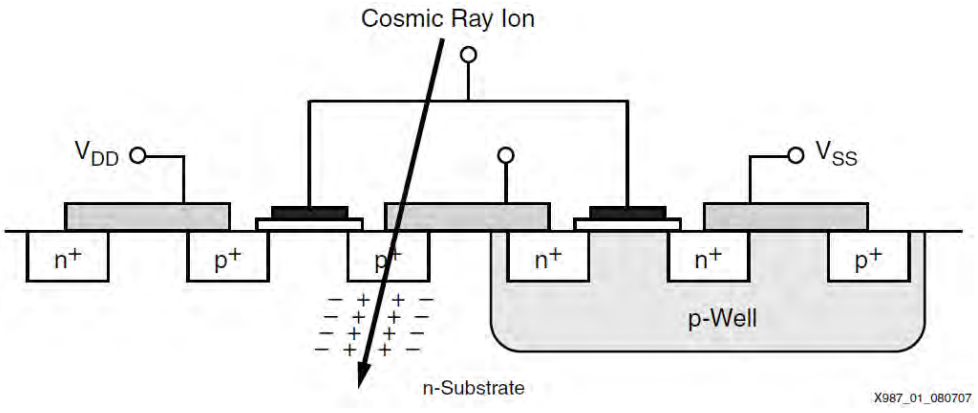


Figure 5. Effects of an ionization event on a transistor inside a typical FPGA (from [12])

Ionization effects on materials can be broken down into three possible effects, Compton scattering (CS), pair production (PP), and the photoelectric effect (PE). The photoelectric effect occurs when a high energy radiation photon is absorbed by an incident atom in the material, the atom responds by emitting an electron in a random direction and the nucleus recoils with an equivalent momentum. This electron can go on to cause other interactions. This primarily occurs with lower-energy photons. As photon energy increases the Compton scattering effect is more likely, where the incident photon transfers some of it energy to motion of the target electron, the photon is then re-emitted at a lower energy and can go on to cause other interactions. At the highest levels of incident photon energy (>1.02 MeV), pair production reaction can occur. In this situation the incident photon interacts with a heavier nucleus and produces an electron and positron with equal but opposite momentum. These two particles then usually go on to produce secondary interactions. These effects are probabilistic; a very high energy photon may cause any one of the three effects. A photon must have a certain threshold energy to cause Compton scattering and must have a minimum of 1.02 MeV to cause pair production. An example of these probabilities, comparing the atomic mass of the target nucleus to the incoming gamma energy, is shown in Figure 6.

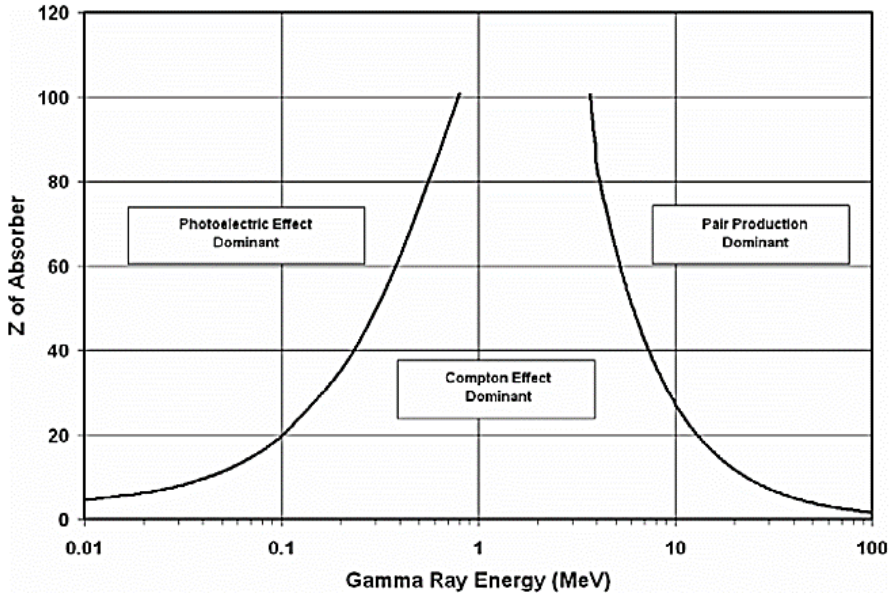


Figure 6. Illustration of gamma ray interactions with various types of material. Areas of dominant interaction types are indicated (from [13])

Total ionizing dose (TID) is a measure of the accumulated damage due to ionizing events, measured in rads, occurring within a device or component. In metal oxide semiconductor (MOS) devices, this damage ordinarily takes the form of a buildup of trapped charge within the thin oxide layers of a device [14]. As an ionizing event occurs in the silicon oxide (SiO_2) layer, it generates a trail of electron/hole pairs (EHPs). The freed electrons are highly mobile and quickly, on the order of picoseconds, migrate away based on the existing electric fields in the area. The holes left behind move in the opposite direction but migrate much more slowly by swapping electrons with nearby atoms. This process causes an immediate shift in the threshold voltage of the device. Depending on device temperature, it can take seconds for equilibrium to be restored. As they migrate through the material, some of these holes become trapped at the SiO2/Si interfaces, and this leads to a permanent voltage shift in the device. The energy of the incident radiation affects the number of EHPs formed, and the electric field strength affects their ability to recombine, as shown in Figure 7. The more EHPs that escape immediate recombination, the greater the overall damage to the device

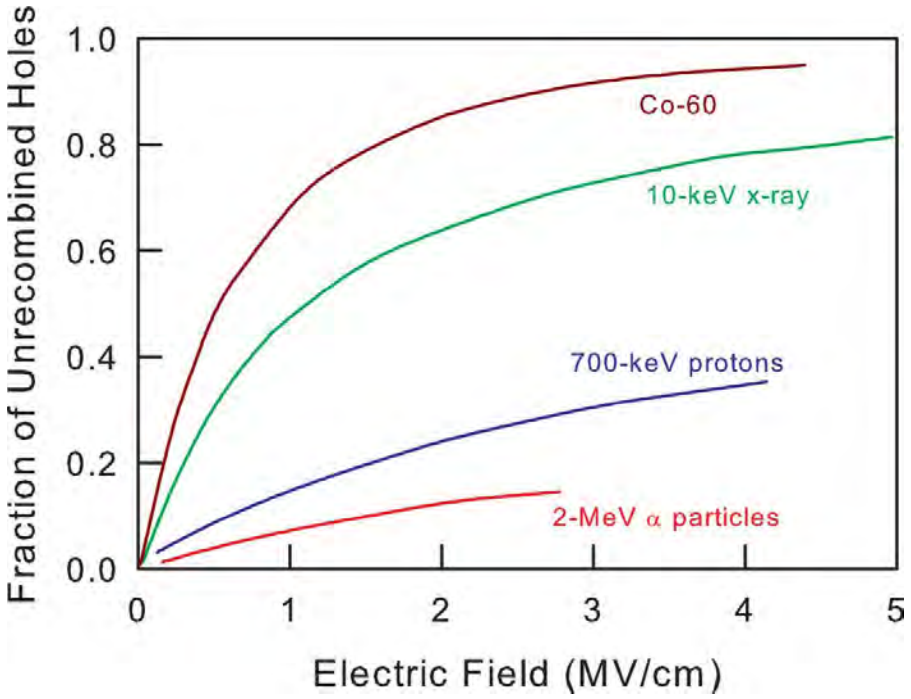


Figure 7. Effects of the electric field strength and the incident radiation intensity on recombination of EHPs (from [15])

Displacement effects are generally caused by two effects, either Rutherford scattering or direct nuclear interactions. In the case of Rutherford scattering, an incoming charged particle undergoes an electrostatic interaction with the target atom. This can lead to an ionization effect, or a displacement effect, where the target atom is physically moved from its location in the crystal lattice. The primary means of energy exchange are via the electrostatic fields of the particles and not physical absorption. If contact does occur, for example an inelastic collision between a neutron from the radiation field and a silicon nucleus in the electronic device, then a nuclear interaction has occurred. This impact can also displace atom from their location in the crystal lattice, and this movement often causes secondary interactions. Other nuclear events, such as a protons being absorbed by material nuclei and subsequently producing more alpha particles when they decay, are also considered displacement effects.

Is it important to note that these events and their causes are closely intertwined, and the secondary effects produced can be significant. A single interaction that generates a displacement event may produce several secondary displacement events before the energy is dissipated. Each of those secondary events may produce other ionizing events. Similarly, a high-energy ionizing event may also cause secondary displacement effects or ionizing events. In addition, if these effects occur in a semiconductor device with a current applied, the electric field generated can impact the path and final disposition of the resulting particles. This last interaction and the permanent damage caused by displacement effects are what generate the undesired effects inside semiconductor devices.

3. Radiation Effects on Electronic Component Operation

Exactly how these various events affect the operation of the device itself depends largely on the construction of the device, exactly where on the device the event occurs, the energy released by the event, and the state of the device at the time of impact. If the event produces a measureable effect, it can be considered a single event effect (SEE). Physically, these effects take the form of current or voltage spikes within the device. Soft errors are seen as temporary transients that can be corrected by resetting or cycling the

power to the device. Hard errors occur when the SEE causes permanent damage to some portion of the FPGA. As we cannot readily measure the voltage or current at every point inside an operating FPGA, we can only classify these events as they relate to the operational logic of the device. These operational errors can be tracked and further broken down based on the type of effect they have on the FPGA.

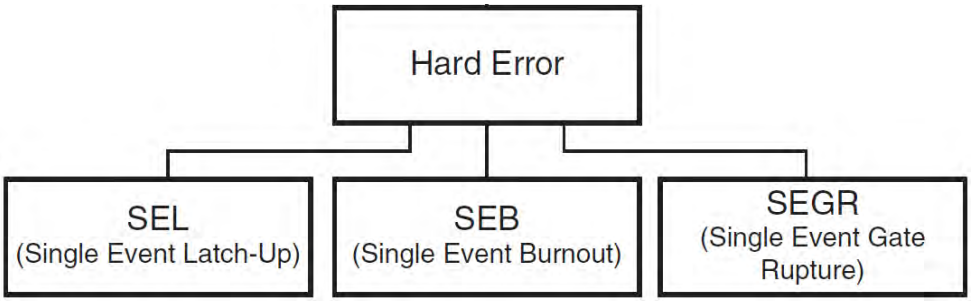


Figure 8. Types of non-recoverable single-event effects. (from [16])

The susceptibility of a device to hard errors is of significant importance in this discussion. If the FPGA suffers permanent damage quickly in the operating environment, then it becomes useless in short order, regardless of any fault-tolerant logic designs. The three major types of hard errors include the single-event latch-up (SEL), where a SEE causes a transistor within the device to become "stuck" in a particular state, the single-event burnout (SEB), which applies mainly to power metal–oxide–semiconductor field-effect transistors (MOSFETs) and results in a permanently forward-biased transistor, and the single-event gate rupture (SEGR), a rupture of the gate's oxide insulation, which generally occurs only with high energy SEEs.

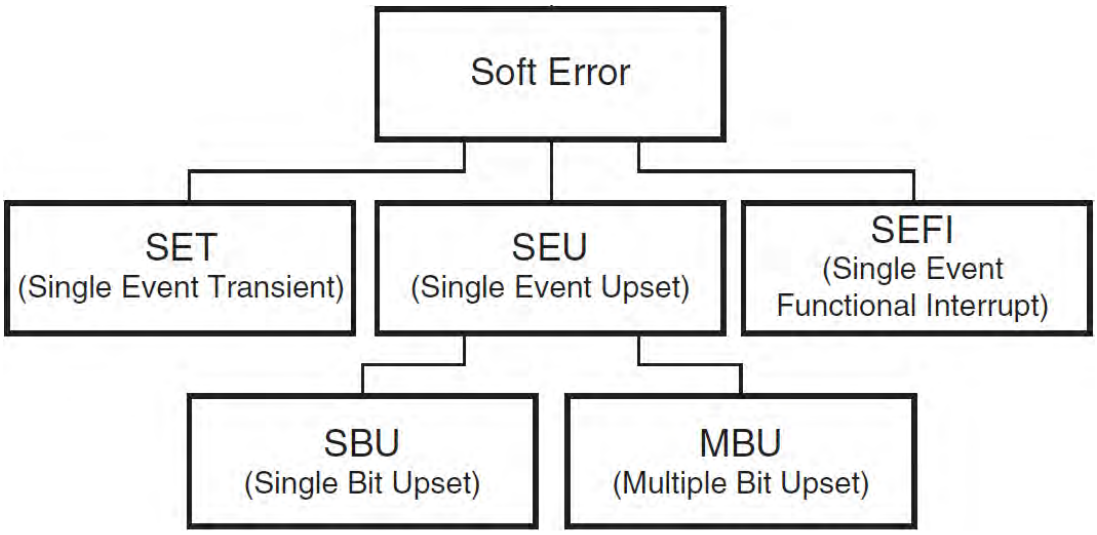


Figure 9. Types of recoverable single-event effects (from [16])

Soft, or recoverable errors, are much more common than hard errors and are seen in almost all FPGA or ASICs. These generally fall into one of two major categories: single-event transients (SETs) and single-event upsets (SEUs). An SET occurs when an SEE causes a voltage or current transient in the FPGA. If this occurs at the wrong time, for example on a clock edge where the logic is currently checking for a value, then an erroneous signal may be propagated into the logic function. An SEU is more specific, and occurs when the SEE causes a "bit flip" in the FPGA. A flip-flop, memory element or latch within the FPGA must be changed from zero to one or vice versa by the SEE to be classified as an SEU. If multiple bits within the FPGA are impacted by the SEE, this is classified as a multi-bit upset (MBU).

A third category, the single-event function interrupt (SEFI), is sometimes used to describe a major disruption to the FPGA that requires reconfiguration or power cycling to restore operation. A SEFI usually occurs in the configuration or support sections of the FPGA, such as the configuration memory or the JTAG interface [12].

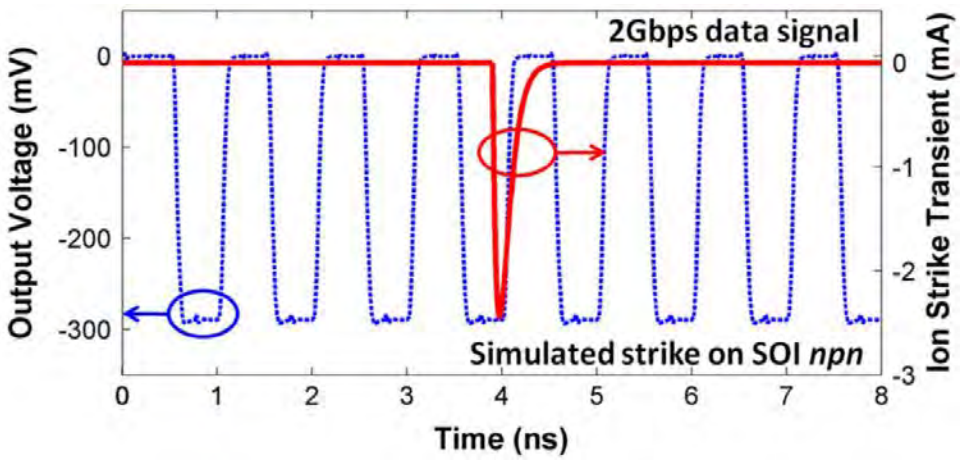


Figure 10. Simulation of a single-event transient occurring on a clock edge (in red), causing a missed transition of a flip-flip (in blue) (from [17])

The radiation conditions that satellites must operate in are quite varied in the type and relative strength of the flux encountered. Every satellite sees wide ranges of conditions from cosmic rays, solar storms and sun spots, and variations in the earth's magnetic belts. For any FPGA operating under these conditions, it is not a question of if these SEEs cause errors, but how frequently these errors will occur. Some design features and methods can reduce the probability of errors, and other features try to limit the effects of the errors. In any case, the errors will occur and must be properly accounted for in the overall design.

B. FAULT TOLERANCE BY RADIATION HARDENING

Reducing the radiation induced error rate in FPGAs and ASICs is a priority for semiconductor manufacturers. Several methods have been successfully employed, from changing the composition of the base materials used in manufacturing to changing the size and shape of features on the silicon. Each of these methods have costs in terms of performance or capability of the device. Due to their more limited demand and more complex manufacturing, these devices also come at a much higher dollar cost per unit. FPGAs can also be more susceptible to radiation effects than ASICs. The internal structure of an FPGA is reconfigurable, and this additional flexibility adds additional locations and modes of failure as the reconfiguration matrix and memory can also be affected.

1. Silicon on Insulator and Silicon on Sapphire

Silicon-on-insulator (SOI) technology is a method of manufacturing semiconductor devices using a layer of electrical insulation, typically silicon dioxide, to separate the bulk silicon layers. This additional layer of material reduces the charge collection volume of each transistor, makes the devices much more resistant to SEUs, and immune to most types of SEL [18]. Much of the industry drive to improve this technology is based on SOI's lower parasitic capacitance, which improves power consumption for handheld devices. As a result, many newer devices, such as the Xilinx Virtex-5, use SOI in their standard products, providing an improvement in radiation hardness for no additional cost. Silicon on sapphire (SOS) is similar to SOI but uses a silicon film grown onto a sapphire (AL_2O_3) wafer. SOS devices are considerably more resistant to radiation [19] but come with a much higher price tag and considerably fewer options on the market.

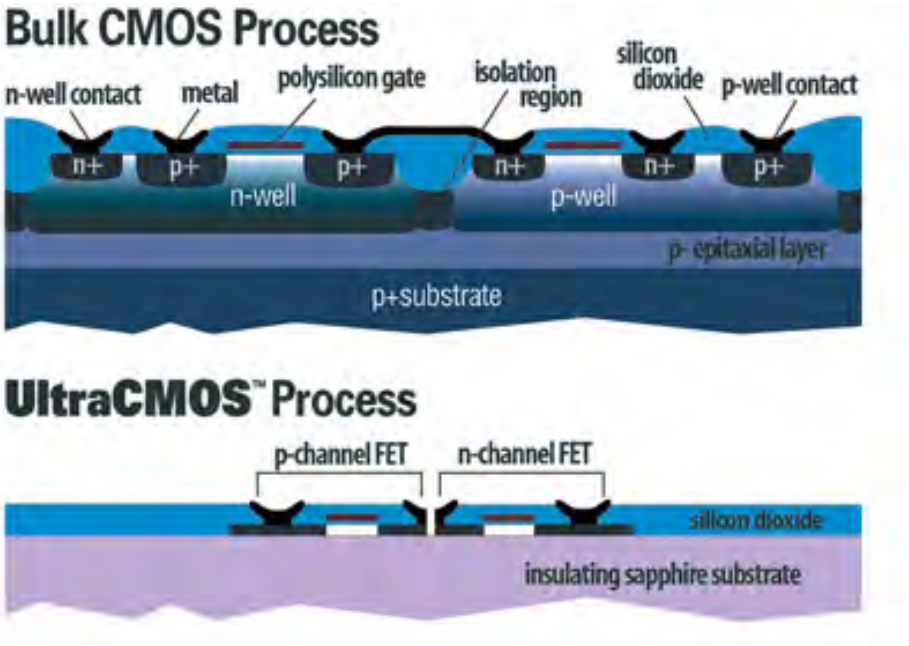


Figure 11. Example comparing a bulk silicon transistor to Peregrine Semiconductor's UltraCMOS[4] (SOS) process (from [20])

[4] UltraCMOS is a registered trademark of Peregrine Semiconductor, Inc.

2. FPGA Technology

The development of programmable logic devices (PLDs) began in the mid-1970s as engineers looked for ways to overcome the significant non-recurring engineering (NRE) costs of building an ASIC [21]. These devices were needed for two primary tasks. One, provide a means of prototyping and testing a design before mass producing an ASIC. The significant cost of creating a single ASIC made this a major cost saving step for any project. The second task was to serve as operational hardware for small-scale production. Even today it is not cost effective to make small quantities of custom ASIC devices. This makes FPGA technology ideal for space applications, where frequently only one or two vehicles are made the same. Early PLDs and FPGAs had low clock speeds and a limited number of gates and interconnections. Modern FPGAs, such as the Xilinx Virtex-7,[5] offer over two million logic cells, 68 Mb of on-chip memory, and can handle 2.9 Tb/sec of I/O via 1200 I/O pins [22]. This capability allows an engineer to put the capabilities of entire computer systems into a single chip [23].

An FPGA consists of several components manufactured onto a common piece of substrate material. This integrated circuit device is configurable via a series of programmable switches connecting the internal logic blocks. An example of a typical FPGA can be seen in Figure 12. Each of the configurable logic blocks (CLBs) contains a series of lookup table (LUTs) and multiplexors (MUXs) that can be assembled and programed per the user's specifications [24]. Each CLB is interconnected via an underlying matrix, and these matrix connections are also programmable to allow connection between CLBs, input/output (I/O) pins on the FPGA package, and other internal components such as memory and built-in clock devices. Each component is susceptible to radiation and each exhibits different responses to SEEs. The interconnection matrix itself can also be susceptible to SEEs and must be considered during design. There are three common categories of FPGAs that are used in aerospace

[5] Virtex® is a registered trademark of Xilinx, Inc.

applications that are differentiated based on the methods used to program their interconnection matrix. These types are anti-fuse, static random access memory (SRAM) based, and flash based.

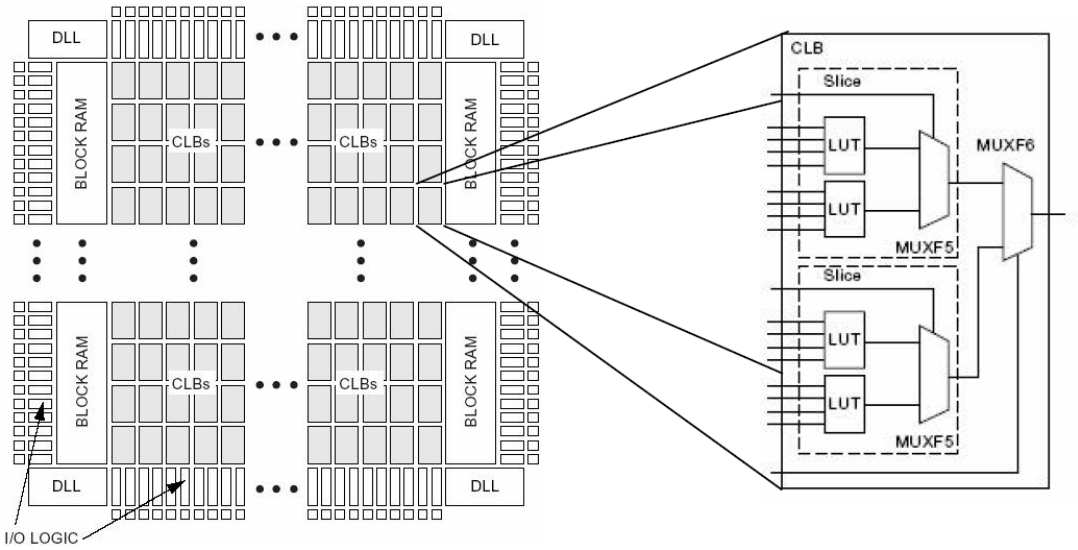


Figure 12. Typical FPGA internal structure (from [24])

a. Anti-fuse FPGAs

Anti-fuse FPGAs are named as such due to the nature of their internal switch matrix. A regular fuse opens when a sufficiently high current is passed through it. An "anti-fuse" circuit closes when a higher-than-normal current is passed through it. Using a programming device, the desired internal switches are permanently fused closed by high current pulses. The switch matrix in anti-fuse FPGA is one-time-programmable (OTP). Once the FPGA design has been programmed, it cannot be changed. This design method offers considerably less flexibility than other methods, but the anti-fuse FPGA does offer a switch matrix that is largely immune to SEUs and TID effects [25]. The CLBs and other internal structures are still susceptible to radiation effects. The one-time-only aspect of this type of FPGA makes them less desirable in many space applications, as they cannot be reconfigured to correct problems or take on new capabilities.

Table 2. Comparison of FPGA switch technologies (from [25])

Switch	SRAM	Flash (EEPROM)	Antifuse
Switch Control	Volatile memory	Non-volatile floating gate NMOS	Non-volatile metallic link
Re-configuration	Fast	Slow	Not available
	Operation V_{CC}	High voltage (20V)	
	Unlimited times	Limited times (~1000)	
	Re-configurable data processing	Off-line	
SEU	Switch very sensitive to SEU	Switch not sensitive to SEU	Switch immune to SEU
TID	Switch has CMOS TID	Switch has typical Flash TID	Switch immune to TID

b. Flash Based FPGAs

Flash-based FPGAs use a switch matrix formed of floating gate transistors. An internal circuit called a charge pump controls the current to each of these switches. Their configuration is non-volatile and stored in a NAND or NOR-type internal flash memory and thus is maintained during power cycling [26]. Flash-based FPGAs can be reprogrammed multiple times while installed in the system. However, the charge pumps are susceptible to TID effects and degrade over time in a radiation field. Typical flash memory structures are NAND based, and these memory structures are also sensitive to TID based damage [27].

c. SRAM Based FPGAs

The last type of FPGA considered here is the SRAM-based design. Similar to the flash-based version, the configuration for this FPGA is stored in memory. In this design, the configuration memory is stored in SRAM switches. A non-volatile memory, either on or off-chip, holds the configuration bit stream until the device is powered on. The internal switch matrix is typically a multiplexor controlled by the configuration memory. This has the advantage of a larger number of possible configurations as well as thousands of reprogramming cycles. The configuration memory

itself is volatile, and the device must reprogram itself at power-on. SRAM based devices are susceptible to SEUs in both the logic blocks and the configuration matrix.

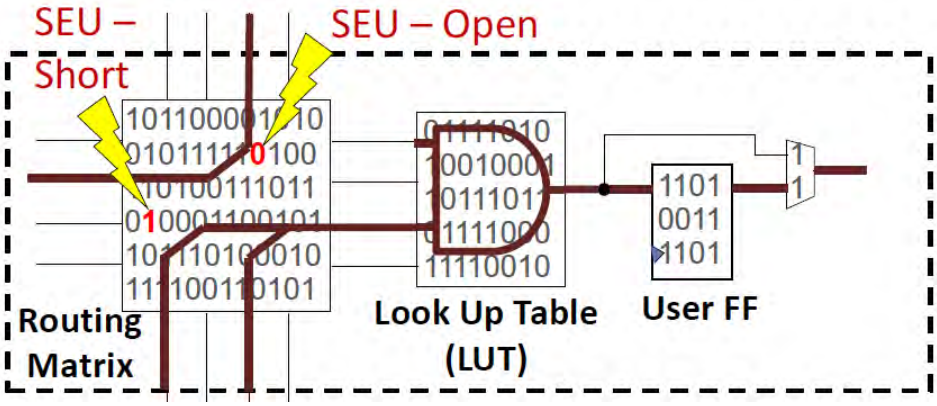


Figure 13. Example of short and open SEUs in an FGPA switch matrix (from [28])

To increase their radiation resistance, manufacturers add redundancy and logic design features to the SRAM memory of some of these devices. Xilinx for example, offers a single-error correction, double-error detection (SECDED) capability for the configuration memory that is built into the FPGA [29] for their Virtex-5[6] and newer series of FPGAs. This capability allows the device to automatically correct any single-bit memory error on boot and configuration and detect and report an error of two bits. These additional logic features, combined with the radiation hardened physical structures, make these "space grade" FPGAs extremely resistant to SEUs but also extremely expensive.

SRAM based FPGAs have additional circuitry requirements that can raise the deployment cost of both the rad-hard and normal aerospace designs. These FPGAs require secondary circuitry on the associated PCB to monitor for errors and provide a means of power cycling to restore the base configuration in case of an error. Older designs required even more external devices such as DRAM, voltage regulators, and oscillators. Newer FPGA devices have all these features internal to device, reducing the requirements. Despite these drawbacks, SRAM based FPGAs are still very popular due to their extremely high feature density, flexibility, and high switching speed.

[6] Virtex® is a registered trademark of Xilinx, Inc.

C. FAULT TOLERANCE BY LOGIC DESIGN

There are a number of fault-tolerant methods that have been developed to improve the reliability of computer systems and logic devices. These designs have been used any time a very high reliability system is required, such as life-critical medical devices or devices controlling nuclear reactor safety systems. The objective of these designs is to ensure that no single fault can prevent the proper operation of the overall system. There are many variations of redundancy logic fault tolerance, including triple-modular redundancy (TMR), dual-modular redundancy (DMR), triple-interwoven logic (TIL), and quadded logic. Fault tolerance can also be provided with reconfiguration based methods, such as scrubbing [30]. Regardless of the specific method or combination of methods, there are three basic goals they all share:

- No single point of failure can cause a system failure
- Faults can be isolated to the component that produced the error
- Faults are contained to prevent propagation

While every method improves the overall reliability of a given logic design, there is a cost in terms of performance. A simple duplication design consumes more than twice the FPGA resources than a non-redundant design. The more complex methods can consume nearly four times the resources. This must be taken into consideration early in the design process to ensure sufficient FPGA capacity is available.

1. Quadded Logic

One of the earliest forms of redundancy was proposed by a Bell Telephone employee named J. G. Tyron in 1958 [31]. His concept, called "quadded logic," took a simple logic design and radically increased the fault tolerance by replacing every gate with four similar gates, using a cross-connection scheme to interconnect each gate. These cross-connects are critical to ensure fault tolerance. An example for a simple half-adder is shown in Figure 14. This concept is resistant to SEUs, with errors being self-corrected between stages. Just from this simple example, it is clear that the minimum price for this type of fault tolerance is four times the FPGA resources.

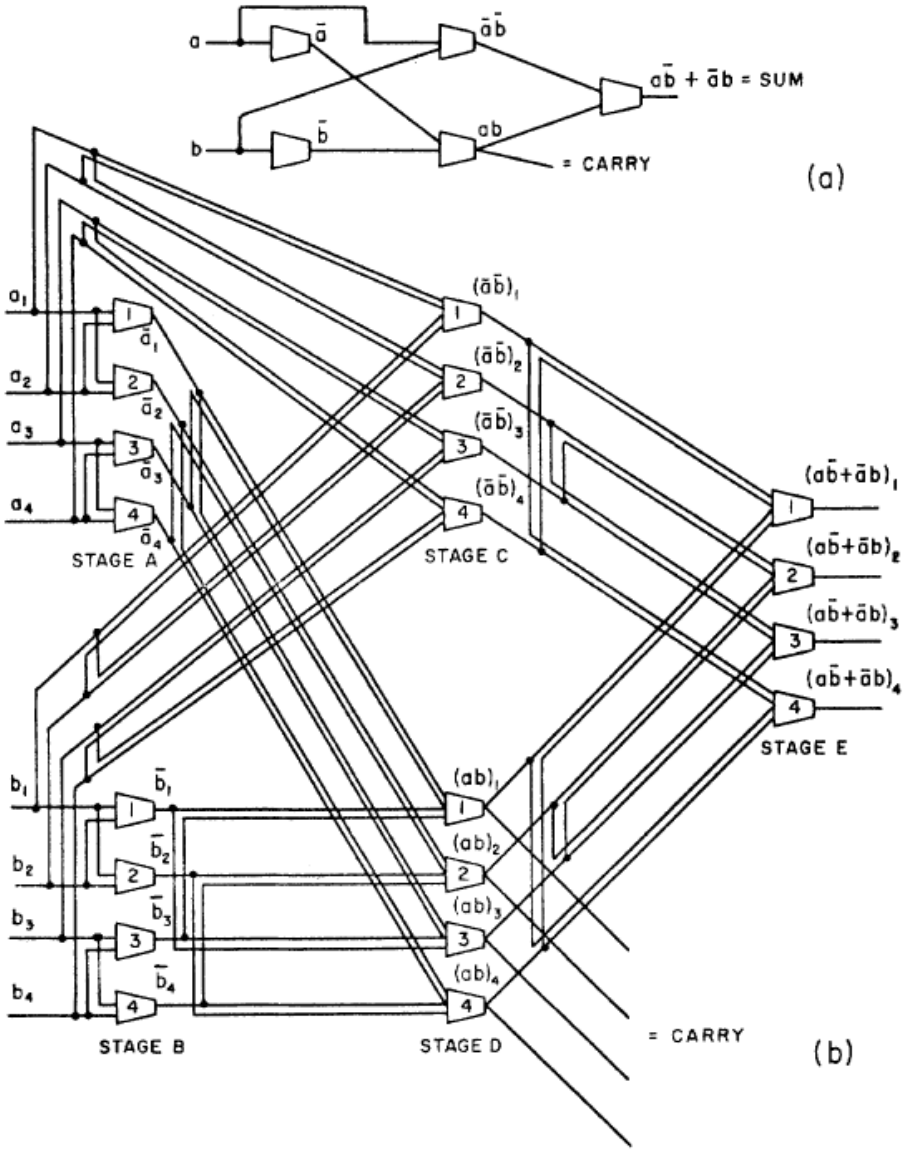


Figure 14. NOR-based half-adder (a) and quadded logic version (b) (from [32]).

2. Quadruple Force Decide Redundancy

Quadruple force decide redundancy (QFDR) is a modified version of quadded logic with some minor variations. In particular QFDR can be applied to logic structures more complex than simple gates. By using a "force decide" layer between logic sections, it can detect when one of the four inputs is different from its neighbors and force an error bit that is carried to the following layer, where the decide step selects the inputs from the

non-error containing logic [30]. QFDR has the advantages of being scalable for logic structures much larger than single gates but still incurs the high FPGA resource penalty of quadded logic.

3. Triple-Modular Redundancy

Improvements in reliability of computers via an arrangement of redundant circuits and voting was initially developed by J. Von Neumann in 1965. The output of a series of three identical logic functions is sent to a voter circuit, which he called the "majority organ" [33]. The voter compares the three inputs and provides an output based on a majority function, as shown in Figure 15. The number of inputs to a voter is odd to ensure that the result is unambiguous [34]. This basic design has a serious flaw, in that an error in the voter circuit could result in an erroneous output signal. To account for this issue, the voter circuitry itself is usually replicated as well, which can be seen in Figure 16. Even in this case, some final determination must be made to determine the majority for the three outputs.

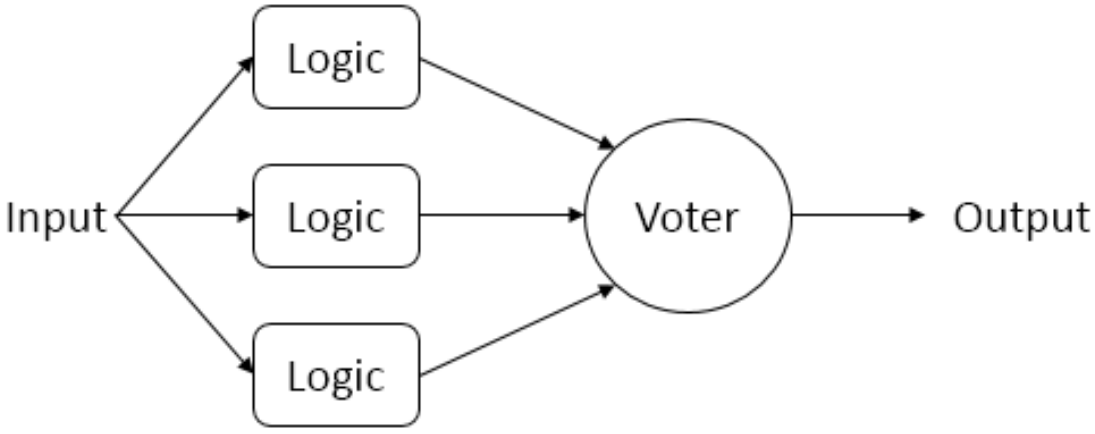


Figure 15. Basic concept for triple-modular redundancy

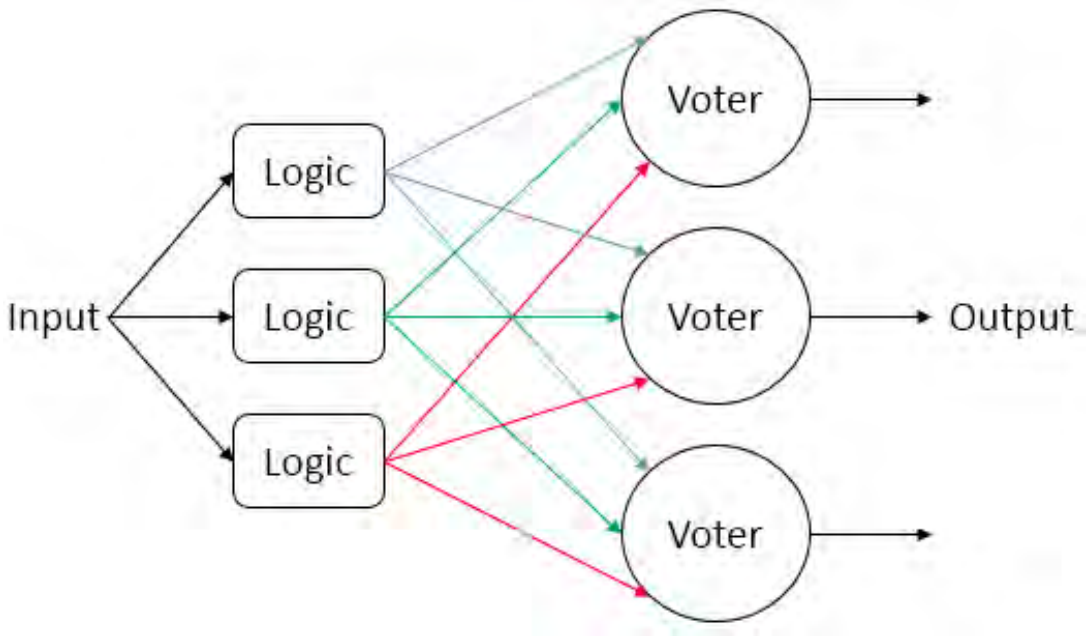


Figure 16. TMR with triplicated voter circuits.

Use of TMR in an FPGA can take several forms based on the needs of the designer. Block TMR (BTMR) is the simplest form and consists of a single voter circuit for triplicated blocks of logic. This method is no different from the basic TMR methods that have just been described, except the logic blocks may be fairly complex functions. As the complexity of the blocks increase, this method becomes less useful for real world application. Errors in more than one block simultaneously become more likely as the block size increases, and with no feedback function to "reset" a block after an error, the system may soon be corrupted [35].

Local TMR (LTMR) is an improvement on BTMR. In LTMR, a set of flip-flops (FFs) are placed between the output of voter circuits and the next logic block. In this case, the correctly voted value is sent to each FF in that layer. This method provides a feedback feature and ensures that errors are not propagated past any given logic stage [35], however, this method has drawbacks as well. In LTMR the individual clock and reset/clear lines for the FFs are not triplicated and can cause potential errors if an SEU occurs on one of those signals. In addition, the FFs themselves add additional complexity and increase the sensitivity of the design to SETs.

A further improvement in reliability was introduced in the form of global TMR (GTMR). Similar to LTMR, the logic blocks are divided by FFs. In GTMR, however, all the clock generators and other asynchronous inputs such as clear signals are triplicated. This method has the greatest requirements for FPGA resources and introduces new potential problems if the clocks between the modules are not properly synchronized. The final method discussed here, distributed TMR (DTMR), attempts to correct this problem by triplicating all the elements but uses a global clock and reset signal [35]. Regardless of the specific TMR method used, there are significant challenges in verification of the final FPGA design to ensure all critical elements are properly triplicated.

4. Triplicated Interwoven Redundancy

Triple interwoven redundancy is another form of fault tolerance and is formed from a combination of some quadded logic concepts with TMR. In this method, the initial logic design is triplicated, and the resulting outputs are cross-connected in a similar fashion to quadded logic. The design must first be broken down to a gate-level and inputs and outputs classified as critical or subcritical based on the gate types. For example, a stuck-at-zero fault is critical for a NAND gate but subcritical for a NOR gate. Then the gates are interwoven at each state to ensure that a critical output is a subcritical input for the next stage. Ideally, as the error progresses through the logic, it is eventually "healed" and disappears. An example of the half-adder discussed earlier but produced via a TIR method is seen in Figure 17. This method has proven to be comparable in terms of fault tolerance to TMR methods [36].

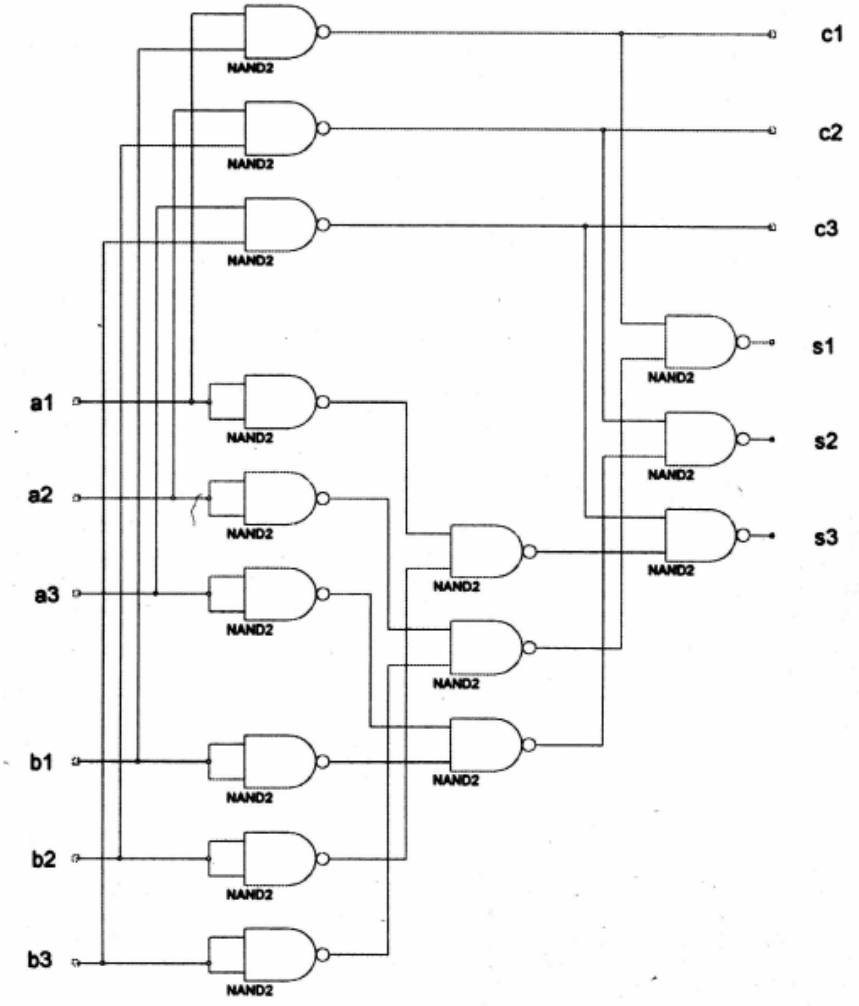


Figure 17. A NAND based half-adder with triple interwoven redundancy (from [36])

5. Error Correcting Codes and Reduced Precision Redundancy

Two other fault-tolerance methods used in digital logic systems are error correcting codes (ECC) and reduced precision redundancy (RPR). In an ECC system, such as a Hamming code, a mathematical algorithm is performed on a bit stream or memory segment, such as a group of registers [37]. This method is used to detect and correct single bit errors by adding redundant bits and setting allowed bit patterns. If the actual pattern does not match one of the allowed patterns, it can be corrected by changing the offending bit. Another fault protection method is known as reduced precision redundancy (RPR). In an RPR system, two less-precise versions of the original circuit are operated in parallel [38]. For example, if the original circuit was a 64-bit

mathematical processor, an RPR version may be a much smaller 32 or 16-bit system. The outputs of these three parallel processes are compared, and if an error in the precise version is detected, the lower precision solution is used, giving a "good enough" solution rather than an outright error. If the error is in one of the two lower-precision results, then the precise result is deemed to be correct. This method can be employed internally to an FPGA for arithmetic processing tasks but not for general logic [6]. In the case of the NPSCuL sequencer, neither of these methods is appropriate. The sequencer has a very limited memory storage, making ECC protection impractical. As the sequencer operates only as a simple sequential machine, it has no complex processing, and as such, a reduced precision version is not possible.

6. Configuration Scrubbing

Configuration scrubbing in an FPGA is performed when the configuration memory is periodically refreshed with a "golden copy" of the memory. This resets the device to a known-good state and clears any SEUs that may have occurred in the configuration memory. In order to be effective, the configuration memory must be error-free. Many manufacturers use ECC storage for the memory to provide SECDED features for the memory itself [25]. The scrubbing can also be performed any time an error is detected in the logic output, but this requires additional logic to detect the errors internal to the logic design.

A complete scrub of an FPGA renders it inoperable until the scrub is completed. The device must then be restarted and allowed to operate. In the case of mission-critical systems, this necessitates multiple redundant FPGAs, driving up the cost of the system. To avoid this, an alternate method termed "readback and compare" [39] is implemented. In this method, the configuration memory is first read, then compared to the reference copy. The re-write of the FPGA is then only required if an error is detected. Another alternative is partial reconfiguration, where only certain sectors or portions of the FPGA are reconfigured, leaving the other portions operating [40]. This method is complex and requires very detailed planning during the place-and-route step of FPGA planning to ensure the logic used is properly located.

D. CHAPTER SUMMARY

In Chapter II, a general background of fault tolerance for FPGA devices and why it is required was provided. The sequencer that is discussed in following chapters will operate in a harsh radiation environment. While on orbit it will encounter radiation at varying levels from every direction. Despite this, the sequencer must operate correctly and any faults must not affect the overall operation.

When a semiconductor device, such as an FPGA, is operated in a radiation field, there can be a wide range of potential effects. Soft and hard errors can be categorized, and those can be further broken down into the various types of SEEs such as SEUs, SELs, and SETs. Each type can have a detrimental effect on the routine operation of an FPGA. Long-term exposure, in terms of the TID, must also be considered when determining the performance of a particular device. Different manufacturing techniques and materials can reduce these effects significantly.

Knowing that an FPGA will experience these events to some degree during operation, we must discuss how to mitigate the effects. This is where the various forms of logic come into play. Proper implementation of the different forms of TMR can make a device nearly immune to SEEs. Combined with periodic and situational FPGA scrubbing, this can make for a highly reliable device.

The technology and principles discussed here are critical to the development of almost any electronic device destined for operation in a spacecraft. The mission of the spacecraft determines the radiation conditions it encounters, and the performance requirements and budget drives the parts selection. In very high radiation environments, even rad-hard devices may experience SEUs. In low-cost applications such as NPSCuL, selecting the largest and most radiation hardened silicon may not be an option. Proper employment of the fault tolerant techniques discussed here is critical for proper operation of designs like the NPSCuL sequencer that is discussed in detail in the next chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SEQUENCER DESIGN

A. SEQUENCER HISTORY

When the NPSCuL program was initiated, the engineers had two possible concepts for control of P-POD door opening. The first option was opening via direct control from the host spacecraft. This concept minimizes the need for additional hardware and control in the NPSCuL system. The burden to monitor and control the opening of each P-POD door could be handed off to the robust flight computer of the launch vehicle [1]. While this concept is attractive for NPS, due to the reductions in cost, local testing, and development, it comes at a cost to the launch vehicle. Relying on host control shifts more of the testing burden to the owner of the host spacecraft. Given that the host spacecraft typically operate under a program that is orders of magnitude larger than the NPSCuL program, this may not be acceptable. Even if the host spacecraft's program did accept such a requirement, they would likely require a completed unit for testing early in the testing cycle [41].

The second option for control of the P-POD doors is an independent control system, contained entirely within the NPSCuL itself. This concept uses an on-board controller, known as a sequencer, to control the sequence and timing of P-POD door opening. The sequencer also handles any required telemetry data [41]. The sequencer receives redundant electrical power from the launch vehicle. When powered on, it commands the non-explosive actuators (NEAs) that actually open the P-POD doors, deploying the CubeSat using a spring. A micro switch attached to each P-POD door allows for monitoring of the door position by the sequencer or host spacecraft.

For the first flight planned for the NPSCuL system, the launch vehicle was unable to provide all the required power and data lines, and an NPS-designed sequencer was not going to be available within the expected flight delivery timeline. Therefore, a commercial product, designed and built by Ecliptic Enterprises, was to be used for the mission. This sequencer was designed for integration into an Atlas-V launch [42]. Ecliptic Enterprises worked with both NPS and ULA to ensure proper integration with

the NPSCuL and the Atlas V's Centaur upper stage avionics. The CubeSats selected for the first flight were not ready on schedule and so the first NPSCuL flight was cancelled.

The sequencer is an essential piece of the entire NPSCuL system, serving as the primary control system. If the sequencer fails to operate, the entire mission of the NPSCuL fails, as none of the installed CubeSats are launched. If the P-PODs open without sufficient delay between deployments, the CubeSats could collide. The NPSCuL is an auxiliary payload on a larger mission. In a worst case scenario, a failure of NPSCuL could endanger the primary mission spacecraft. This, above all other requirements, must not be allowed to happen.

B. REQUIREMENTS

The design requirements for the sequencer discussed here are consolidated from a variety of sources. Some requirements such as the specific dimensions allowed for the hardware mounting are taken directly from NPSCuL design documents [43] [10]. Others are drawn from datasheets of components selected as possible candidates for use in future designs. Still other requirements were generated following discussion with Dr. James Newman and Dr. Herschel Loomis during the course of design.

1. Operational Requirements

Operational requirements for the sequencer answer the basic question of "what does it do"? These are the primary considerations for all other design questions. The sequencer must meet the following requirements to meet the mission objectives of the NPSCuL system:

- Provide a fully programmable launch sequence for eight P-PODs. The sequence and timing of individual P-POD launches is reprogrammable.
- The delay between P-POD launches can be set from one second to one hour in one second intervals.
- The launch sequence commences automatically at power on. A programmable delay from power on to initial launch is provided but no additional launch command is needed.
- No hold/pause feature is required. To halt the launch, the system must be powered down.

- The system must be fault tolerant. SEEs and other system faults must neither cause an inadvertent launch nor prevent an expected launch.
- The launch sequence and timing can be reprogrammed at any time up to a few hours prior to launch. Reconfiguration capability after launch is not provided.

2. Electrical Interface Requirements

The sequencer designed for this application has three primary electrical interfaces. Primary and redundant electrical power is supplied from the host spacecraft. Primary and redundant power from the sequencer is supplied to each P-POD to actuate the door itself. Finally, each P-POD has a door switch, which allows the sequencer to determine whether the P-POD door is closed or open.

a. Power Supply

Typical host spacecraft operate using a 28 V DC electrical distribution bus. The completed sequencer operates at this voltage. Specific peak and idle current values are available until final hardware selection. An estimate of 8 A peak and 2 A idle is used here with the following assumptions:

- A typical NEA draws 6 A when actuated (additional detail provided in following sections)
- No more than 1 P-POD NEA are actuated at a time
- The internal power supply converts the incoming 28 VDC power to the 5.0 V, 3.3 V and other voltages required for the FPGA and host adapter.
- The power supply provides the necessary voltage regulation and adequate thermal and electrical protection for the sequencer.

b. Non-Explosive Actuators

The electrical actuators for the P-POD doors consist of a small spring-loaded arm held in position by a material with a low melting point. This material is surrounded by an electrical heating element. When sufficient electrical current is supplied, the material softens, and the spring actuates the arm. This arm then mechanically unlatches the P-POD door. An example can be seen in Figure 18. These

devices each have unique operating curves that delineate the time for actuation based on the supplied current. For this operation, the maximum actuation time of 120 ms [1] is used.

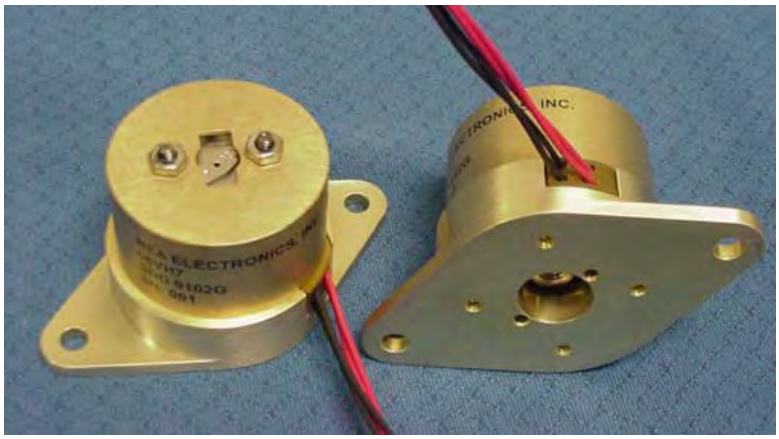


Figure 18. NEA model 9102G non-explosive release mechanism (from [1])

c. Door Position Detection

Two simple micro switches are mounted to the P-POD door mechanism. The switches are connected such that when the P-POD door is closed, the circuit is closed. When the P-POD door opens, the switch actuates and the circuit opens. This switch is connected to a 3.3 V or 5.0 V power supply connection and simply provides a binary signal regarding the position of the P-POD door [43]. While this signal is used to indicate the successful door opening of a P-POD, it must be noted that there is no onboard mechanism to determine if the CubeSat actually deployed properly. The sequencer only has indications of door position. The purpose of the switches is to validate that any given door does not open until it is commanded to do so. Although a failure resulting in an open door, and therefore CubeSat deployment, prior to launch is considered unrealistic, it is prudent to monitor the doors so that in the event of a launch anomaly, the NPSCuL system can be exonerated or implicated, as appropriate.

3. Mechanical Requirements

The final design for the sequencer must fit within the volume of the NPSCuL splitter auxiliary device (SAD) enclosure. The launch vehicle for the second planned flight of NPSCuL had sufficient resources to provide eight primary and redundant power and eight data lines. Therefore, all that NPSCuL had to provide was a splitter auxiliary device (SAD). The original version of the SAD is shown in Figure 19. This original version provided a pass-through connection for power and control from the host spacecraft's flight computer to the individual P-PODs on the NPSCuL. This enclosure mounts directly to the side of the NPSCuL structure. The second version of the SAD incorporated a PCB mounted to the bottom to replace the wiring harness bundles and to simplify the manufacturing of the harnesses.

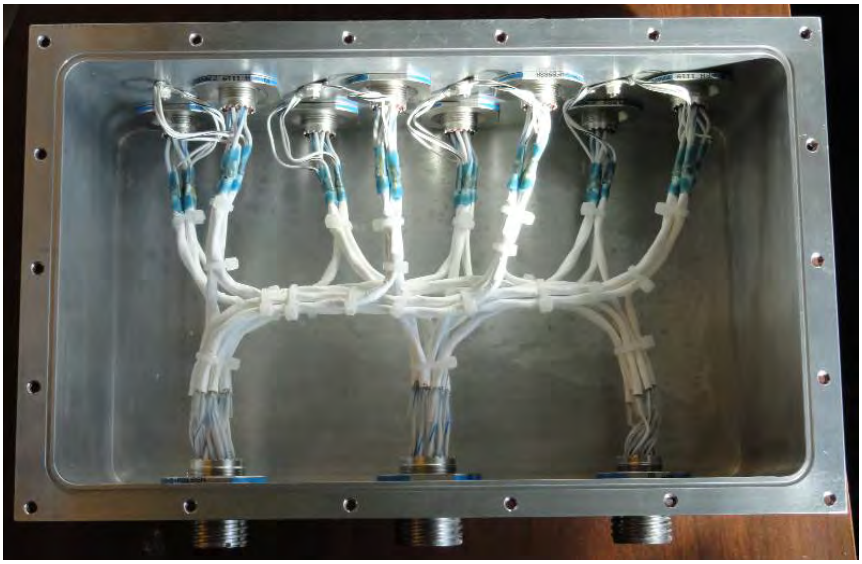


Figure 19. NPSCuL splitter auxiliary device with wiring harnesses (from [2])

A newer version also incorporates a PCB to replace the wiring harnesses and provides additional space to mount the FPGA, power supply, relays, and other electrical components. The design for this PCB can be seen in Figure 20. This enclosure provides external interface connectors for both primary and auxiliary power for the sequencer's power supply. These eight connections supply power to the NEAs and door position switches. For the purposes of this thesis, the specifics of mechanical mounting, sizing,

and issues such as thermal control are not discussed. This requirement is only used for rough estimates of required PCB space for a proposed final design.

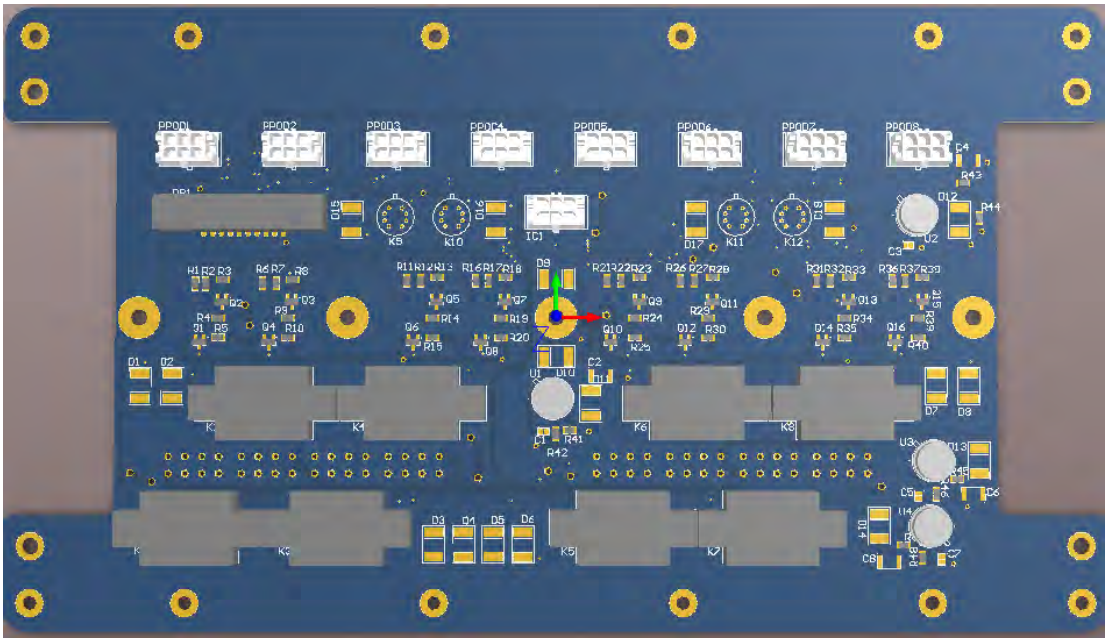


Figure 20. SAD with PCB design, showing mounting and connector location (from [2])

4. Performance Requirements

The initial performance requirements for the sequencer are fairly simple and easy to accomplish. The device must operate quickly enough to execute the assigned launch sequence. A typical launch sequence is set to allow two or three minutes between launches. A launch command must be applied to the NEA for a minimum of 120 ms to ensure a successful launch. As there are no limiting high-speed clocking requirements, a design decision was made to select the lowest possible clock speed since SEUs have been demonstrated to occur more frequently as clock speeds are increased [44]. Operating at the lowest clock speed necessary to meet these requirements provides maximum reliability.

C. SEQUENCER FLOW AND STATE MACHINE

The simplest method of meeting the operational requirements listed above is a simple state machine using sequential logic. A method using a pre-existing design, such

as the NPS configurable fault tolerant processor (CFTP) [3], was briefly considered but would be significantly more complex than this application requires, and that complexity brings along an increased susceptibility to SEUs.

1. Flowcharting the Design

The first step taken was to incorporate the operational requirements into a flowchart for design. This flowchart is seen in Figure 21. A number of design decisions were made at this point for the sequencer.

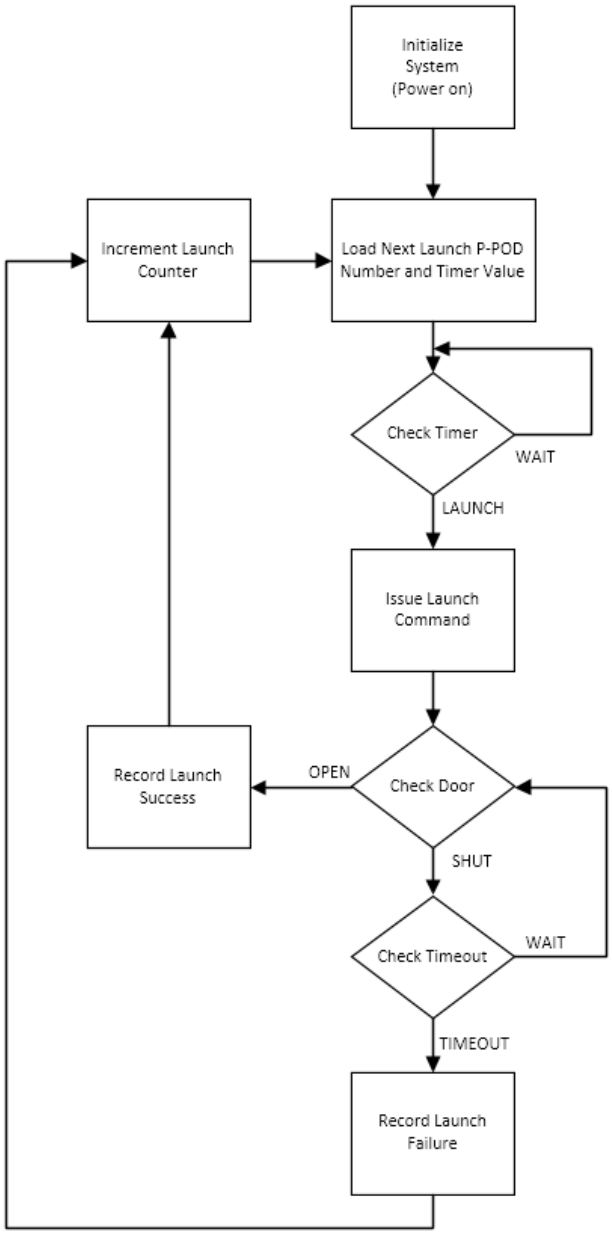


Figure 21. Sequencer flow chart

The first decision was to make use of the P-POD door status signal. For the purposes of testing, as well as possible future use in reporting telemetry to a ground station or host system, a launch status is recorded for each P-POD launch. When a launch command is issued to a given P-POD, the door status is checked continuously. If the door is reported as "OPEN," the launch is recorded as a success. If the door status does not report "OPEN" within a set timeout period, then the door is assumed to be stuck

or the NEA failed. In this case, the launch is recorded as failed. The available sensors on the P-POD do not detect a failure where the door only opens partially, the CubeSat jams on launch, or if the launching spring fails. In each of these failure cases, the sequencer still records a successful launch.

The second important design decision is the use of a separate "memory" to store the launch sequence and desired launch delays. To maximize the simplicity of the system, this sequence and delays can be hard-coded into the final design and integrated directly into the logic. This limits the complexity and makes the system that much more resistant to logic failures; however, this conflicts with the desire to have an easily reconfigurable design. A separate memory within the FPGA reduces the requirement to re-perform significant levels of testing following a change in launch sequence.

2. Developing a State Machine

The next step was determination of what type of sequential machine to use for this implementation. There are two varieties of state machine to consider for this application. The Moore machine or the Mealy machine. In a Moore type machine, the output logic function of the circuit depends only on the current state. In a Mealy machine, the output logic function uses both the current state and the current input to determine an output. In this sequencer design, the only external input is the P-POD door status. As this input does not directly impact the output to the NEAs, a Moore machine was chosen.

The final selection entails eight states for the P-POD launch states and five states to record success or failure, start the sequence, advance the sequence and a wait state. With a total of 13 states, a four-bit state machine is required. With 16 possible states, measures must be taken in design to ensure the unused states are compensated for to prevent an unknown state condition. With this initial concept completed, development of a useful software model could begin.

D. SOFTWARE DESIGN

Using the state machine developed, we created a series of functional blocks or modules. The design goals at this stage in development were to create a series of modules that

would meet the functional goals of the sequencer while being adaptable to one of the TMR techniques discussed in Chapter II of this thesis. This provides a means of verification and comparison between a functional base design and a design with fault tolerant logic features. It must also be a design that can be used for a comparison of manual and software-controlled TMR solutions. A block diagram of this design is shown in Figure 22. The modules created and shown here have the functions discussed in the following subsections.

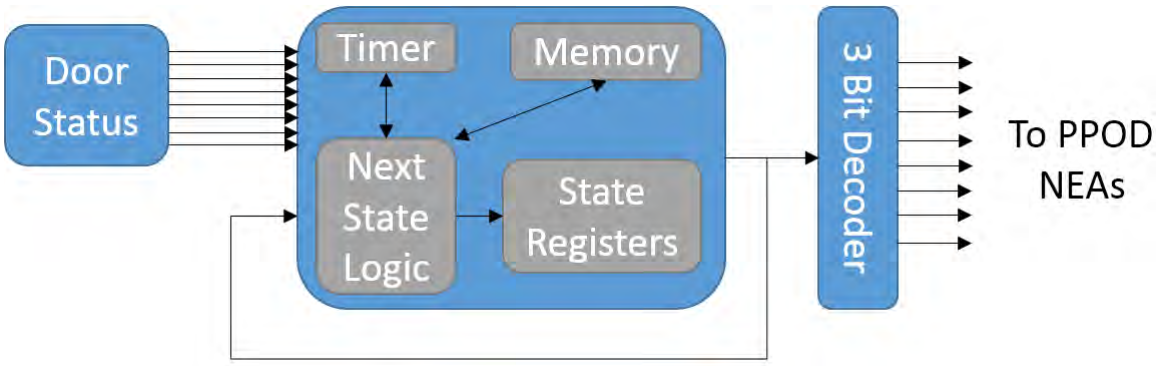


Figure 22. Sequencer block diagram without fault-tolerant features

1. State Registers

The module that stores the actual current state of the machine is known as the state register. In this design, these are formed of a parallel group of four D-flip-flops to provide the required four-bit state information. This module is clocked with the global clock and has an asynchronous clear input that resets the FFs to the starting state. The input of this module is considered the "next state," and the output is the "current state" of the machine.

2. Next State Logic

The next state logic is the "brain" of the sequential machine. It takes the input from the state registers to determine the current state. Combining that information with the input from the P-POD door status switches and the timer and memory modules, it determines the next state of the machine. This is done entirely with combinational logic, leading to a very reliable module.

Using the design developed in the previous section of this chapter, we established a set of inputs and outputs, and each state was developed. A detailed description of what occurs in each state is defined in the following.

a. Start State

This is the initial entry state for power up of the state machine. In this state, the first P-POD number and delay are loaded from memory, sent to the timer, and the timer is started. Following a launch, the machine returns to this state for the next launch. The next state for this step is the wait state for all operating conditions, as seen in Figure 23. In the block diagram, the next-state arc is labelled with its destination state and the outputs that are asserted.

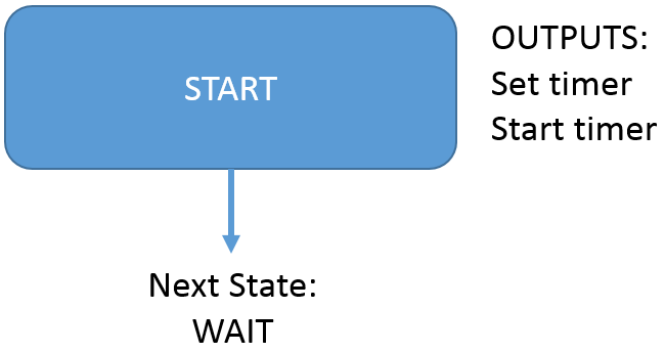


Figure 23. Algorithmic block diagram for the start state.

b. Wait State

In the wait state, the state machine is waiting for the next P-POD launch as seen in Figure 24. The logic first checks the timeout signal. If the timeout has not been raised yet, the machine recycles to the wait state again. If the timeout has been raised, the logic sets the timer for the P-POD door switch delay. The logic then sets the next state to the currently selected P-POD in the launch sequence. For example, if the current P-POD is #4, the next state is the launch state for P-POD #4.

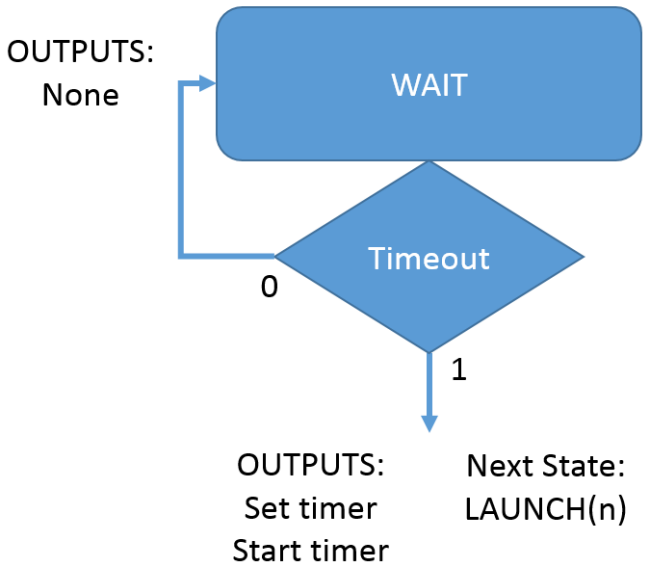


Figure 24. Algorithmic block diagram for the wait state.

c. Launch State

Eight of these states exist in the machine, one for each P-POD. An example block diagram is seen in Figure 25. Upon entering this state, the machine immediately starts the timer for the P-POD door. The actual signal to the NEAs is generated by the launch decoder when it detects the state machine entering a P-POD launch state. The machine then checks the door status of the door associated with the P-POD being launched and the timeout signal. It remains in the launch state until either the door indicates open or the timeout signal is raised. If the door indicates open, the logic assumes the launch was successful and sets the next state to the "launch success" state. If the timeout occurs, the logic assumes the door did not open as expected and sets the next state to the "launch fail" state.

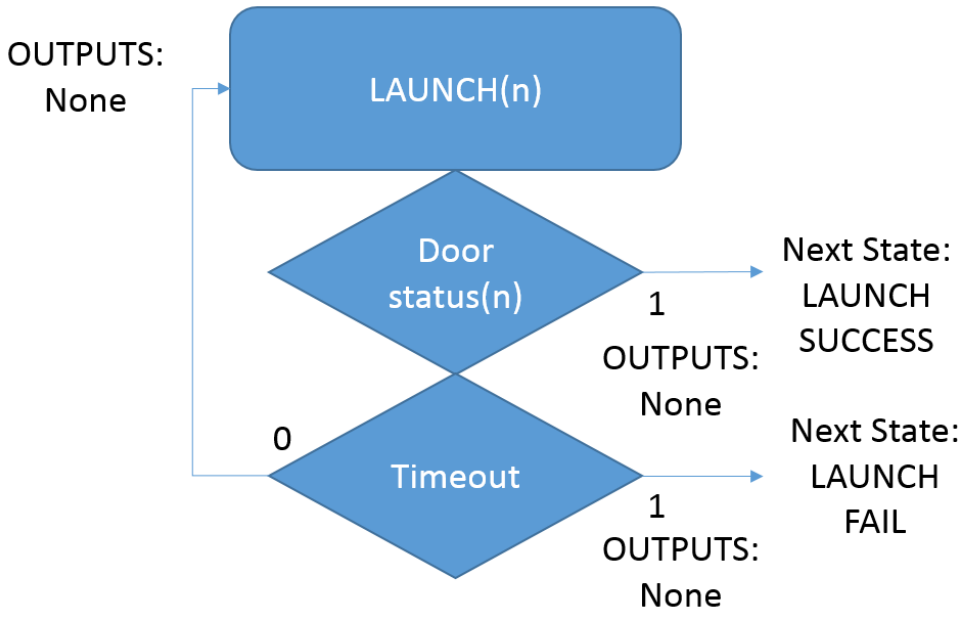


Figure 25. Algorithmic block diagram for the launch state.

d. Launch Success

In the launch success state, the machine sets the associated launch status bit in the memory to one to indicate a successful launch. It then sets the internal advance bit high and sets the next state to the advance state. A block diagram is shown in Figure 26.

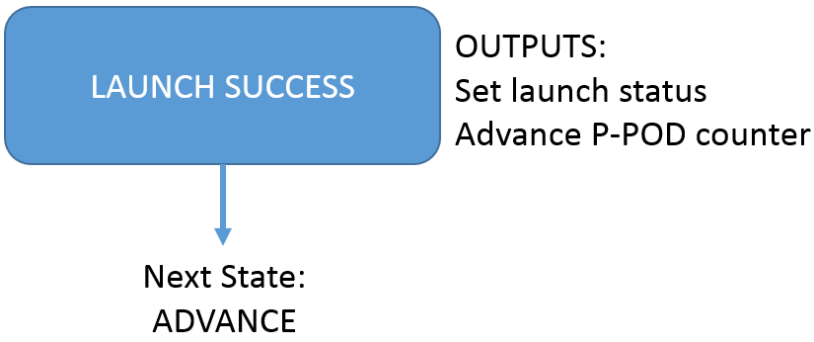


Figure 26. Algorithmic block diagram for the launch success state.

e. Launch Fail

In the launch fail state, the machine sets the associated launch status bit in the memory to zero to indicate a failed launch. It then sets the internal advance bit high and sets the next state to the advance state. The block diagram for this is exactly like that of the launch success state, except the launch status bit is set to zero vice one.

f. Advance

In the advance state, the machine sets up the transition to the next launch state. A check is performed to see if the final P-POD has been launched, as indicated by the address already at seven. If the end has been reached, the next state is set to "done," and the machine operation is halted. If the end has not been reached, the machine advances the address of the current P-POD select lines by one. This serves to load the next P-POD number and associated delay from the memory module. The timer is reset, and the machine next state is set to the start state.

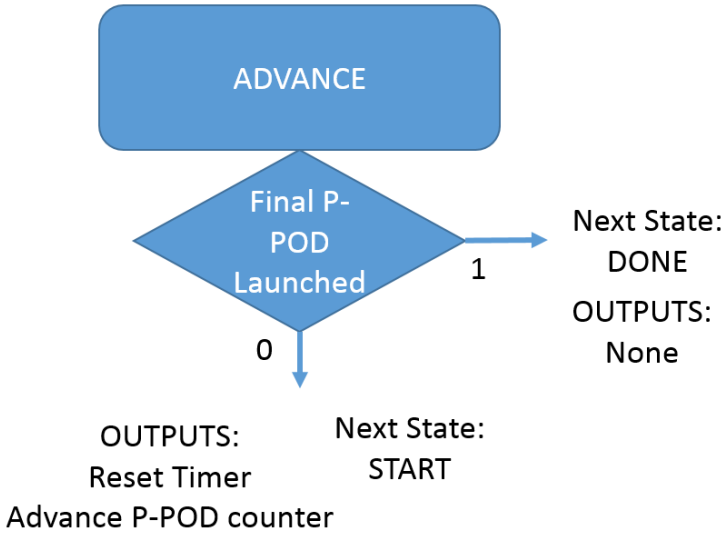


Figure 27. Algorithmic block diagram for the advance state.

3. Timer

The timer used here is a simple counting device. It has a multi-bit input for a time value and a single bit "start" command bit. It also takes input from the global clock and

the system clear commands. The output is a single bit "timeout" value. The timer is "set" with a value from the memory or next state logic and then started. It increments its internal counter on each clock, and when the set value is reached, the timeout signal is raised.

4. Memory

The system memory module stores the launch sequence and required time delays in a simple lookup table. This function was placed into a separate module to allow for simple changes to the sequence and to allow for future changes to how the sequence was stored. It also stores the "launch status" bit for each P-POD, a one for a launch success and a zero for a launch failure. Future revisions can use more complex storage methods and external memory or internal FPGA memory. This module was generated as a separate unit to simplify the future needs of the engineer. To mimic a real memory, this module takes a three-bit address input and a launch status bit input. The module then provides a three-bit number for the selected P-POD and a four-bit number for the programmed time delay for that particular P-POD.

5. 3-Bit Decoder

The final component of the design is the output decoder. This module has a three bit input, and eight individual output connections, one output for each P-POD. This component reads the current state of the machine, and if the machine is in a "launch" state, it generates a high logic signal on the appropriate output pin. This module is nothing more than a standard binary decoder in this application.

6. State Encoding

Selection of state encoding was performed to make the final output simple. This allows the least significant three bits of the state registers to be fed directly to the three-bit decoder to command the NEAs for launch. The state encoding that was employed is described in Table 3. The additional "DONE" state was added from the preliminary state diagram to provide a concrete final state and prevent an error from causing the sequencer to unnecessarily restart the launch sequence.

Table 3. Selected state encoding.

	State Encoding			
State	MSB			LSB
Launch P-POD #1	0	0	0	0
Launch P-POD #2	0	0	0	1
Launch P-POD #3	0	0	1	0
Launch P-POD #4	0	0	1	1
Launch P-POD #5	0	1	0	0
Launch P-POD #6	0	1	0	1
Launch P-POD #7	0	1	1	0
Launch P-POD #8	0	1	1	1
START	1	0	0	0
WAIT	1	0	0	1
ADVANCE	1	0	1	0
LAUNCH FAIL	1	0	1	1
LAUNCH SUCCESS	1	1	0	0
DONE	1	1	0	1
UNUSED	1	1	1	0
UNUSED	1	1	1	1

The unused states in this design must be accounted for during the actual implementation of the design. If an SEU or other error occurs during operation that causes a bit flip in one of the state flip flops, the machine can enter an undefined state and remain stuck there until the entire system is reset. The manual method of dealing with this issue is to assign states to every unused state that simply direct the machine to a known good state on the next clock cycle. For this design, additional states that direct the machine back to the "WAIT" state are appropriate. Using the modern design software however, we can create a "default" state, and the software will automatically correct this problem without the need for multiple additional entries. The complete behavioral code developed to fully describe this machine is found in Appendix B.

E. HARDWARE DESCRIPTION

During the course of design, a potential problem was recognized regarding the output signals to the NEAs. Regardless of the fault tolerance method used in the FPGA, each output signal must still leave the chip itself via individual output pins, and that one

signal wire is a potential point of failure for the overall system. To mitigate this effect, another solution was proposed. The output of the sequencer logic to the P-POD's NEA needs external fault tolerance. A final output voter must be located outside the FPGA itself on the supporting PCB.

1. Hardened Launch Voter

The "hardened launch voter" was created to provide a final fault tolerant step before a launch signal is sent to the P-PODs NEA. This method requires that multiple outputs from the FPGA are fed into a radiation hardened majority voter to produce a final launch command. For this design, a three output design was used but could be expanded if required. A simple schematic can be seen in Figure 28. This design uses radiation hardened, solid state relay components. The Microsemi Corporation offers parts such as the MHS series of five amp relay parts that fit this application very well [45].

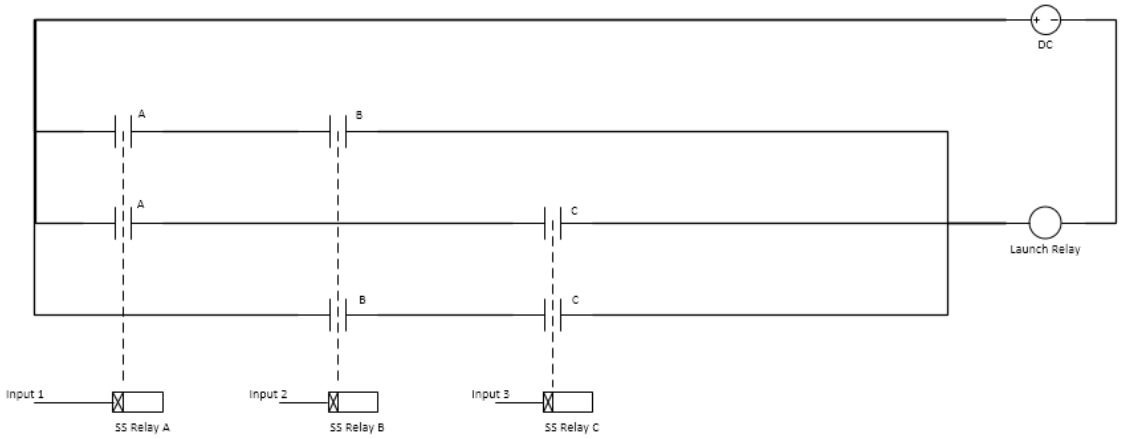


Figure 28. Radiation hardened launch voter

The construction and integration of this hardware into the actual SAD PCB will be performed during future integration. This hardware can provide additional fault tolerance for launch regardless of the particular fault tolerance method used in the sequencer itself.

For the purposes of this thesis, the hardware voter is simulated with a set of simple logic gates. A single output can then be generated and the results for triplicated

modules can be compared. Due to the very robust and radiation resistant design for this hardware, it is very unlikely that any SEEs will occur during operation. In a real application, the FPGA logic experiences significant errors before an error occurred in the hardware voter. For this reason, SEEs in the simulated hardware voter module are not explored.

2. Input Triplication

The only input to the sequencer as designed are eight single bit inputs, one from each P-POD door switch. In most fault tolerant designs, this input requires fault protection as well. In that situation, a redundant series of inputs needs to be obtained from the source. These redundant inputs then require their own voter or verification logic to deal with potential SETs in the input signals and buffers. This step was not taken for this sequencer for these two reasons.

The door status switch input serves an informational purpose only. The sequencer's next state logic decision to launch or not to launch a P-POD is not based on this information. This signal serves only to determine the timeout status to exit a launch state slightly quicker. In the worst case scenario, an SEU in this signal causes a successful launch to be recorded internally as a launch failure. As the signal itself is never used as definitive proof of a successful CubeSat deployment, this should not present a problem during operation.

The signal inputs and outputs into an FPGA are also fairly robust devices. These drivers are much less susceptible to SEEs than other components of the FPGA. Due to the internal construction, several adjacent transistors need to be affected simultaneously to cause an SET in the I/O pins [46]. Given these reasons, and the desire to keep the design simple for manual TMR, the decision was made not to triplicate the inputs for the sequencer. These arguments and discussion points apply just as accurately to real flight hardware as well as the academic scenario.

F. CHAPTER SUMMARY

In this chapter the history and background of the NPSCuL P-POD deployment sequencer was discussed. The two control options were defined, and the importance of the proper operation of the onboard system was explained.

Following this overview, we explored the requirements of the sequencer in detail. The operational requirements were carefully laid out and justified. These operational requirements answered the real question about what the sequencer actually does for the NPSCuL. Additional requirements for both the electrical and mechanical systems are defined and require the sequencer to operate within the existing hardware limits.

Finally, the traditional design process for a sequential machine was applied to this example case. A flowchart was created covering the operational requirements, and a Moore-style state machine was developed from the flowchart. Once this was created, a more detailed breakdown into modules was performed, and specific details were finalized. A design modification to add TMR reliability to the existing SAD Version 3 PCB was proposed, and final design choices were explained. The Verilog software description of the sequencer itself, found in Appendix B, was then developed.

Using the sequencer design developed here, we can now perform testing to determine if this design performs as expected. Software simulations can be created to prototype and refine the design, and the fault tolerance of the simple sequencer machine can be compared to that of a redundant TMR version. Once the operation and performance is satisfactory, the design can be transferred to test hardware for further verification.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SEQUENCER FAULT TOLERANCE

A. SOFTWARE IMPLEMENTATION AND TESTING METHODS

The operation of the sequencer design detailed here is simple to test. A designer must enter the desired launch sequence and delay times into the memory module, then run a behavioral simulation in a software tool to show that the design provides the appropriate output at the appropriate times to the launch hardware. In order to test the fault tolerance of the design, a more complex design is needed. Some means of inserting transient and permanent faults of various types to any potential location within the design is necessary.

1. Software Tools

To implement this design in hardware, the machine must first be translated into a hardware description language (HDL). There are many different types of HDL, but two in particular are the most widely supported and recognized for FPGA design, Verilog and very-high-speed integrated circuit hardware description language (VHSIC-HDL or simply VHDL). The majority of design and testing tools available in the industry support either of these two primary HDLs. Both languages offer similar features, and there is little clear advantage of one over another, the choice primarily being what the programmer is most familiar with. Verilog does have a slight advantage in creating very low-level constructs, down to individual gate level. VHDL tends to perform better than Verilog at very high levels of abstraction, with large system-level blocks. With this consideration in mind, Verilog was chosen as the preferred HDL for this design.

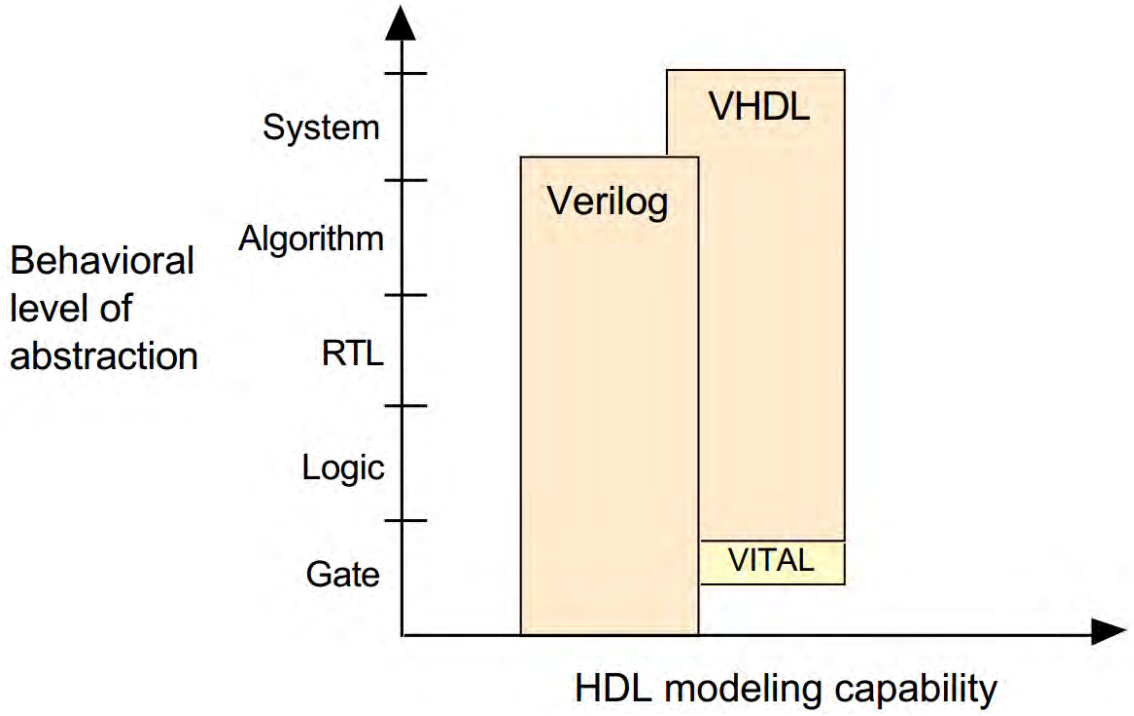


Figure 29. A comparison of Verilog and VHDL considering capability and level of abstraction required (from [47])

The selection of a development environment for the generation and testing of the design is an important choice before moving forward. All of the major FGPA manufacturers, including Xilinx, Microsemi, and Altera offer complete integrated development software packages, all tailored to primarily support their own hardware devices. Following a brief review of the available environments, we selected the Xilinx Integrated Software Environment (ISE[7]). This tool set offers a complete solution for logic design and accepts Verilog descriptions as well as allowing for creation of designs via schematic. Extremely detailed levels of control for all aspects of the design are provided from the initial creation to device programming and testing. The WebPACK[8] version of their ISE is provided as a free download and provides the majority of the functionality of the other paid versions. While Altera and Microsemi offer very similar offerings, the selection was based on one critical factor, which is the behavioral

[7] ISE® Design Suite is a registered trademark of Xilinx, Inc.

[8] WebPACK is a trademark of Xilinx, Inc

simulation software. Xilinx bundles their own product, called the ISE simulator (ISim), into the Xilinx ISE as shown in Figure 30. This simulation software is very important to demonstrate the functionality of the sequencer design. ISim supports mixed Verilog/VHDL, full debug capabilities, and waveform tracing. A single-click recompile/rerun feature allows the user to quickly make changes to the inputs or the design and repeat the simulation. This is a critical capability for the fault tolerance testing that must be performed.

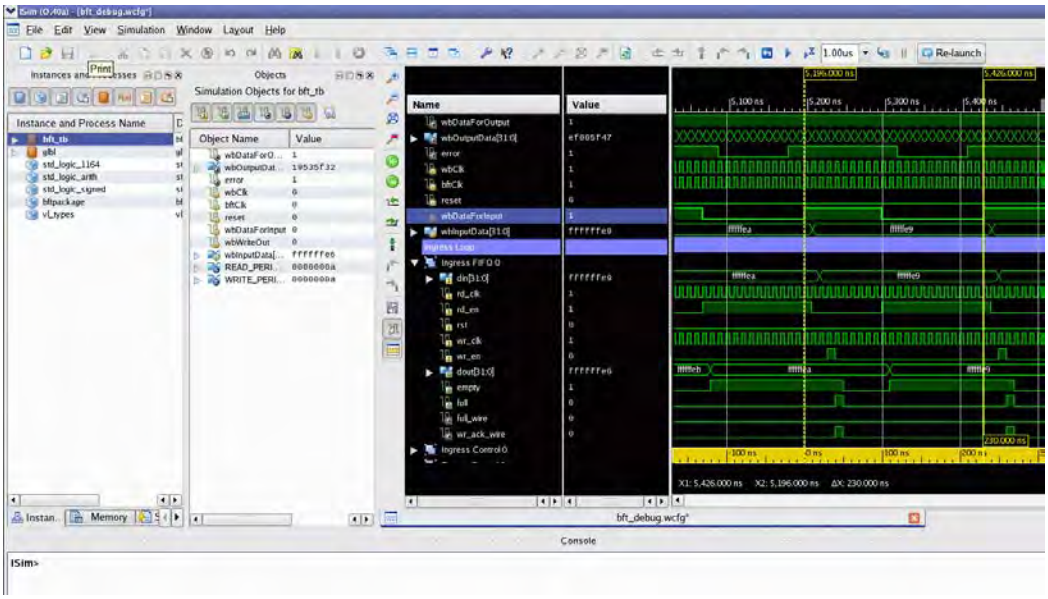


Figure 30. Xilinx ISE simulator (ISim).

2. Testing Methods

Testing the functional operation of the sequencer design is a simple matter. With only eight inputs in the form of door position indications, traditional testing requires testing every possible combinations of inputs and validating the outputs. With eight input bits, this corresponds to 256 possible test cases. This number of cases would be time consuming but easy to perform, but this level of testing does not take into account the multiple launch sequences or delays that can be programmed. This design has 40,320 possible combinations of launch sequence, and each sequence has 16 different possible delays. Doing exhaustive testing requires over 44 quadrillion test cases. While this is

possible with automated tools and scripts, it is time consuming and likely unnecessary. With this simple design, the test cases can be reduced to just the few cases necessary to show proper operation for the expected operating conditions.

a. Potential Error Types

The more important testing is that of the design's performance when experiencing SEUs and SETs during operation. As discussed in previous chapters, these effects can be reduced to three typical effects on the logic of the design.

(1) Stuck-at-Zero. In this error type, the value at a particular gate input or output is stuck at zero, indicating a low value regardless of the required condition. For example, an SEE that caused a short to ground inside one of the device transistors may cause that signal line to remain at zero until a complete reset and reconfiguration of the FPGA occurs. In more extreme cases, such as after considerable radiation exposure, this condition may be permanent and render that gate or FPGA logic block inoperable.

(2) Stuck-at-One. Similar to the stuck-at-zero case, except here the short is to a source that is at the FPGA's operating voltage or simply above the threshold voltage for the individual transistor. This indicates a logical one on the stuck signal line.

(3) Floating Input / Output. A floating input or output is exceptionally difficult to locate or detect during testing of a physical device. An input that is electrically disconnected from a gate input causes the gate to behave erratically. The gate may see the input at a zero or one, or the output may randomly oscillate between the two values. The behavior also may not be consistent, and a floating signal may cause the gate to produce an erroneous output at one time and a perfectly proper output seconds later. This type of error can propagate through the circuit, as a single floating input causes a floating output in the associated gate, which appears as a floating input to the next gate. Unlike a physical circuit in the FPGA, the simulation software can detect these errors and provide a means of locating them. The various simulation packages also provide propagation paths for these errors, making it possible to predict the potential circuit behavior.

(4) Single-Bit Inversions (Bit-Flips). A bit flip occurs when a temporarily stored value in the circuit is altered by an SEE. Some examples are a bit stored in RAM or the current output value of a flip-flop or register. If an SET occurs at the wrong moment in the circuit, then it may also cause a bit flip within the circuit. A bit flip error is more likely to cause a disruption in the operation of a sequential or combinational machine than either of the "stuck-at" type errors. In the case of a bit flip, the propagated value is exactly the opposite of what is intended. In a "stuck-at" situation, that particular line may be stuck at the correct value, masking the error. Due to the transient nature of these errors, they can be very difficult to detect until they have already caused an operational problem in the circuit.

There are several other potential error types in a logic circuit that are not discussed in detail here, including bridging faults feedback bridging faults, "stuck on" gates, and others. The likelihood of these faults and the stuck-at-zero and stuck-at-one errors is very unlikely compared to the possibility of a bit-flip [48]. These other faults occur mainly due to manufacturing defects or other damage to the device from overheating or overcurrent conditions. These effects generally do not occur unless damage occurs to the configuration memory in flash or SRAM based FPGAs. Because of this, the bit-flip and floating logic are the two error types tested. In an operational environment, these effects can occur at any time, at any location in the circuit. Exhaustive testing of each possible effect at every circuit junction is obviously not feasible. These two errors cover both the most likely and worst case scenarios expected to be encountered by the sequencer in operation.

b. Selected Error Sets

A reduced set of fault locations was selected to provide a representative sample of potential errors in the logic. These locations were selected to ensure they have the maximum impact on the operation of the sequencer. Since most SEUs are transient events, the errors inserted are also transient, and the time of their insertion is set to provide the maximum error. The following errors are tested for this sequencer design.

(1) Bit-flip in the Timer Module's Timeout Signal. The timeout signal is used to indicate that the programed value in the timer has been reached. This indicates to the sequencer that the time between launches has expired and it should proceed with a launch or that the wait for the door opening signal has been reached. A bit flip here can cause an early or delayed activation of a P-POD in the first case or an erroneous report of a launch failure in the second case. The first case is tested and timed to try and execute an early launch.

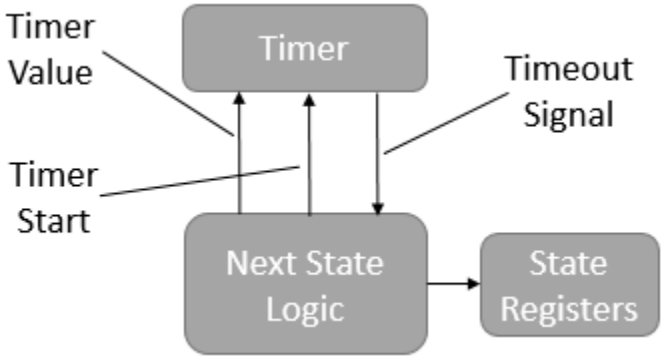


Figure 31. Timeout signal fault location within the sequencer logic.

(2) Bit-flip in the P-POD Select Bus from the Memory Module. The P-POD select bus line is the communication path from the memory module to the next state selection logic. An SET in a signal line here can cause the sequencer to launch the wrong P-POD and result in an incorrect launch sequence. An SEU in the memory itself has the same result. Either error has the potential to cause the sequencer to "skip" a P-POD as well, with the final result being one P-POD launched early and one not launched at all. This error must be inserted at the proper time, when the sequencer is accessing the memory, or it will have no effect on operation.

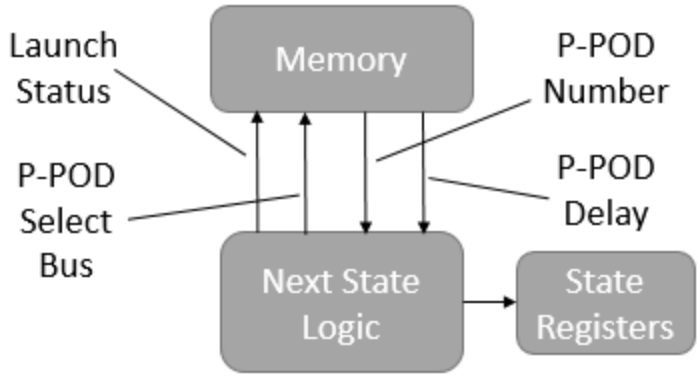


Figure 32. P-POD select bus fault location within the sequencer logic.

(3) Bit-flips in the State output Bus from the State Registers. In any sequential machine, the current and next-state values determine everything that the machine is doing at the time. Any change to these values results in improper and very unpredictable operation. To keep the testing set manageable, the test here is performed on both the least significant bit (LSB) and most significant bits (MSB) of the state variable output. Testing on the state output has the same effect as testing on the state variable input, except it introduces a one-clock delay while the error propagates through the state registers. This error must be present for at least one clock cycle to propagate through the logic but should have no other time or operation dependence to demonstrate a significant change in operation.

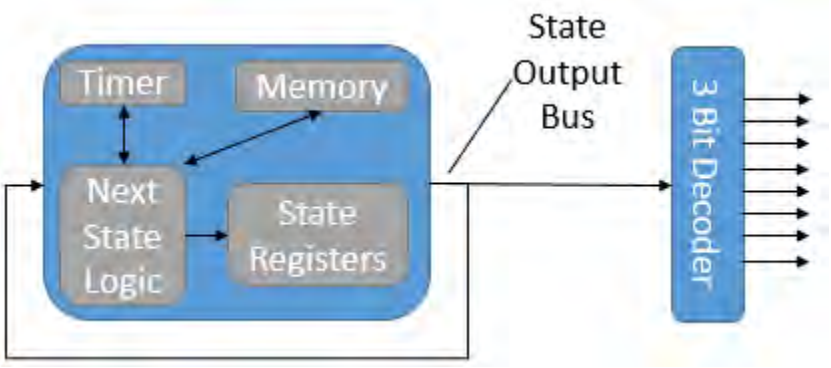


Figure 33. State output bus fault location within the sequencer logic.

(4) Floating Input in the state output Bus from the State Registers. A floating input can be inserted to simply track the potential effects of this error on a real FPGA. This test is performed primarily to determine if the simulation of TMR design is able to handle a floating input.

(5) Bit-flip in the TMR Logic Circuit. When TMR logic is introduced to a design, it also introduces additional locations for SEUs to occur. A bit-flip inserted inside the TMR logic module determines if the design is able to compensate for this particular error. As the manual TMR method used here involves the state registers, the timing of the error insertion is similar to that of the errors inserted in the state variable output.

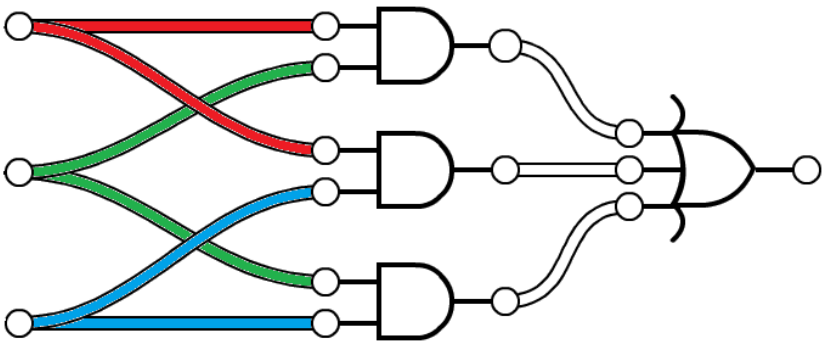


Figure 34. Basic single bit TMR logic circuit.

(6) Bit-flip in the FPGA Configuration. An error in the FPGA configuration memory itself is very challenging to simulate, as the exact effects depend largely on where the errors occur. A worst-case SEU can cause any of the previously mentioned errors. SETs by nature clear from the system shortly after they occur, and SEUs are cleared when the memory is rewritten. An SEU inside the configuration memory, however, could render the affected gate or path in error until the FPGA configuration is rewritten.

c. Configurable Fault Modules

At this point in the design, a means of inserting the desired errors into the circuit was required. The initial testing plan required the use of the simulation fault and

forced-value tools built into the Xilinx ISim software. After initial testing, however, these tools proved very cumbersome for inserting the types of errors that are required. A configurable fault module was developed and created inside the Xilinx ISE. A schematic of this module can be seen in Figure 35.

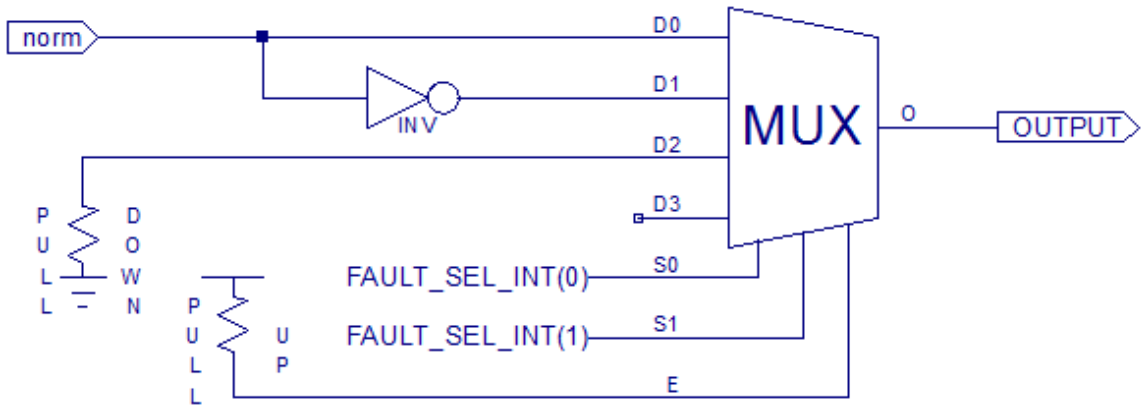


Figure 35. Using a 4-to-1 multiplexor as a fault insertion module.

The obvious first choice was considered of using a simple selectable inverter or two-to-one multiplexor (MUX) for this module. However, this does not allow the testing of a floating input or a fixed zero or one signal. A larger MUX provides this flexibility and reduces the need for multiple types of fault module. The control of this fault module is performed via the two "select" bits of the four-to-one multiplexor used here. The module has four possible modes that can be employed during simulation. In one mode, the module simply passes the input through to the output with no effect on the signal. In the next mode, the module inverts the input as it passes though. The other two modes provide a zero signal and a floating I/O signal. As these modules are for testing only; a SEU fault in these modules or their associated control signals is not considered.

To establish a baseline of operation, a simple launch sequence was selected, starting with P-POD #1, and stepping through each P-POD in increasing sequence to P-POD #8. The minimum launch delay was selected for each P-POD. This reduces both the test length and makes any variations in delay readily apparent. The effect of the door status switches was verified to operate properly. For the testing here,

the doors were assumed to open with no delay, as expected for real-world operation. As such, the door status values were set to indicate open. These values were programmed into the sequence memory module and used for all subsequent tests.

B. SINGLE MODULE PERFORMANCE

An initial simulation was performed to establish the reference condition for error-free operation. The output waveform can be seen in Figure 36.

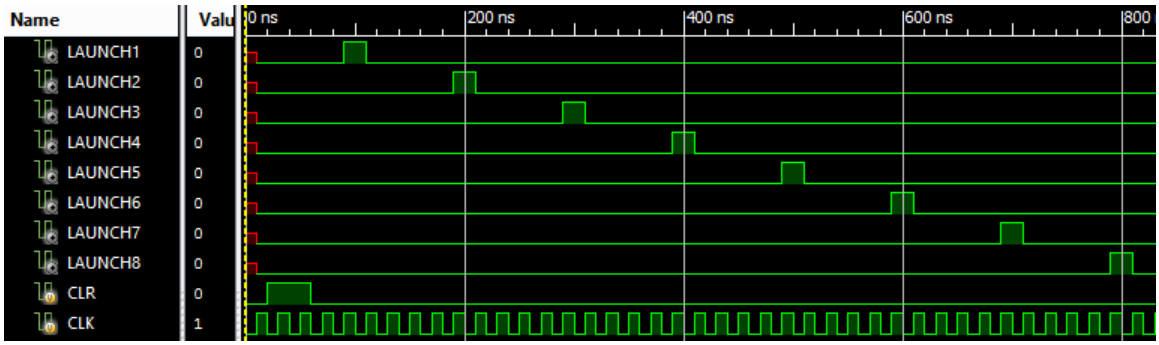


Figure 36. Normal sequencer operation with no fault conditions.

This waveform shows the eight launch signals that send a signal to the NEAs attached to each P-POD. They operate in the programmed sequence from one to eight, with the correct uniform delay between each launch. The length of each launch pulse and the delay is kept to a minimum here to demonstrate the design's function. The design of this sequential machine requires a minimum time in the launch state of one clock cycle. The minimum time between launches is four clock cycles as the machine moves from the launch state to record the status of the last launch, then to load the next launch, wait for the timeout to expire, and finally move to the next launch state. For real world use, these values would be set to those discussed in the sequencer's operational requirements. The clock speed used for testing runs at 25 MHz, or a 40 ns period. A more detailed view of a shorter time period can be seen in Figure 37. The most critical internal signals can been seen here, including the current and next state values and the timer start and timeout signals. The state encoding in decimal with the state names can be found in Table 4.

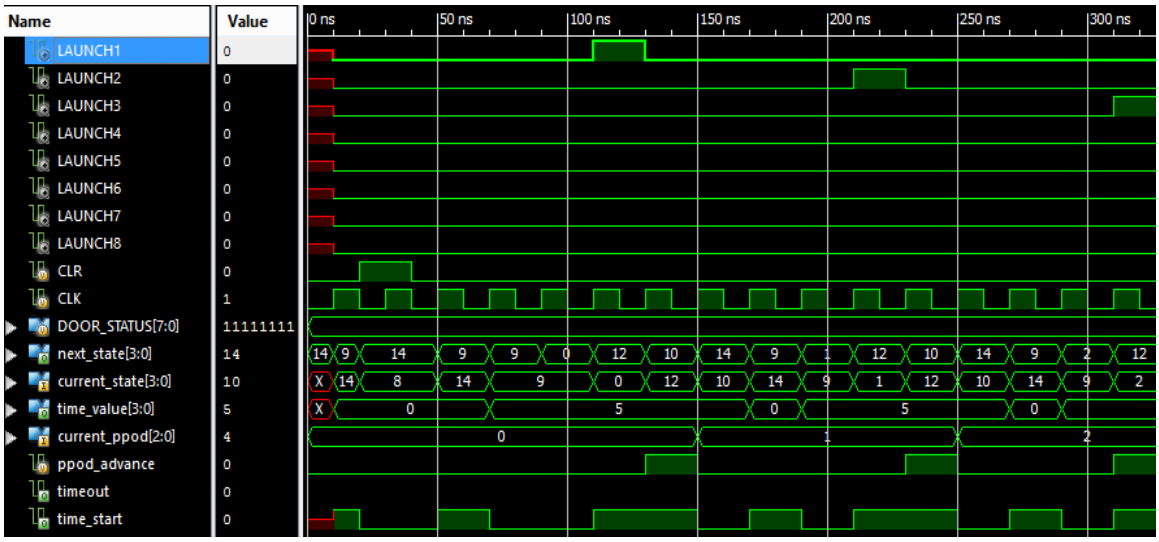


Figure 37. Normal sequencer operation with key internal waveforms visible.

Table 4. Sequencer state encoding with decimal state values.

State Name	State Encoding	State Name	State Encoding
Launch P-POD #1	0	INIT	8
Launch P-POD #2	1	WAIT	9
Launch P-POD #3	2	ADVANCE	10
Launch P-POD #4	3	LAUNCH FAIL	11
Launch P-POD #5	4	LAUNCH SUCCESS	12
Launch P-POD #6	5	DONE	13
Launch P-POD #7	6	START	14
Launch P-POD #8	7	UNUSED	15

1. Timeout Fault

Inserting the fault into the timeout signal results in clearly undesired behavior, as seen in Figure 38. In this test case, a fault was inserted in the timeout signal line at the 260 ns point, and the fault cleared 60 ns later. Analysis of the resulting launch commands shows an unplanned delay before the launch of P-POD #3. In the normal case, this launch occurs at 290 ns, but in this case, it did not occur until 330 ns. For this particular test case, the bit flip error caused a launch delay by effectively blocking the timeout signal. In cases of longer delays, a bit flip could also cause an early launch.

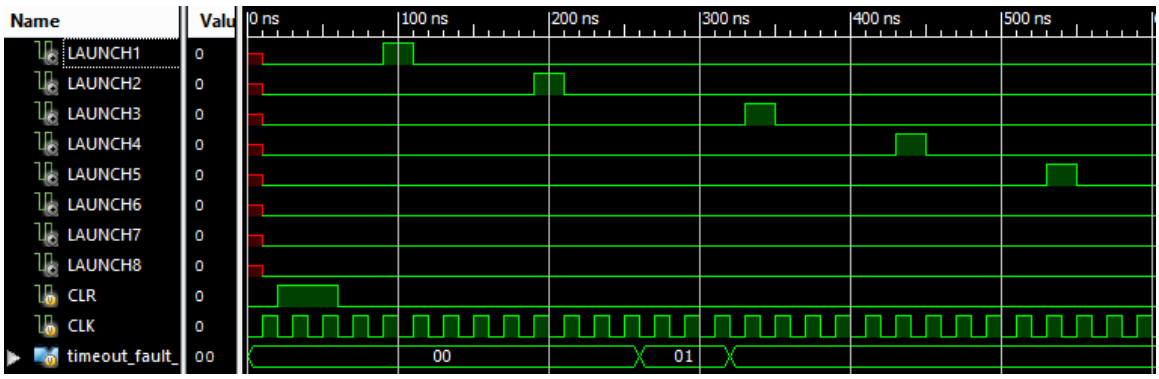


Figure 38. Sequencer operation with a fault in the timeout signal.

2. P-POD Select Fault

A P-POD select is expected to cause significant problems for the sequence if it occurs at the wrong time. The memory returns the incorrect value in the launch sequence, and the logic attempts to launch based on that value. Similar to the previous case, the fault was inserted at the 260 ns point and the fault cleared 60 ns later. In this example, the fault caused the sequencer to select the wrong P-POD, and instead of issuing a launch command for P-POD #3, it instead re-launched P-POD #1. The sequencer then continues on with P-POD #4 and completes the remaining sequence. P-POD #3 is never activated. This would be a mission failure for the associated CubeSats.

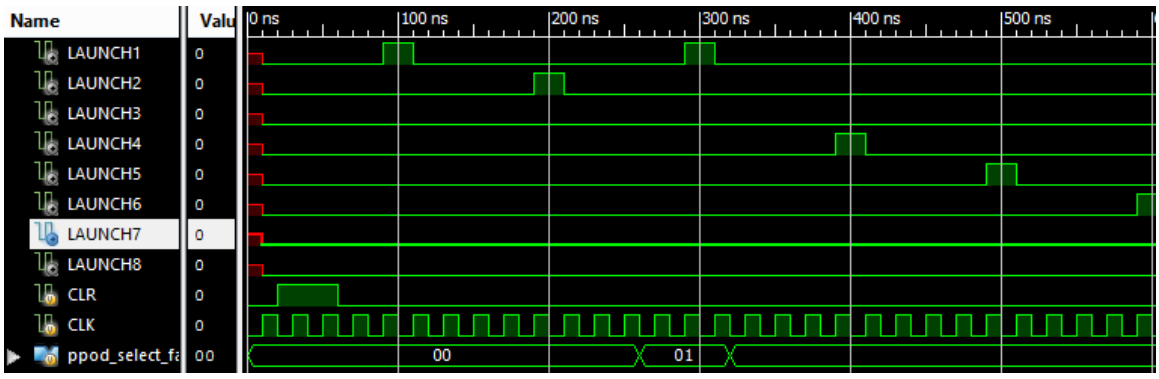


Figure 39. Sequencer operation with a fault in the P-POD select signal.

3. State Output Error—MSB

As is expected for a sequential machine, an error inserted in the most significant bit of the state variables themselves causes a significant series of errors, as seen in Figure 40. The error insertion was performed at the same time as in previous test cases. In this case, we can see that as soon as the error is inserted, the machine immediately enters the launch state for P-POD #1, then one clock period later jumps to activate P-POD #5. After the error clears, the normal operation resumes, but P-POD #3 is never launched.

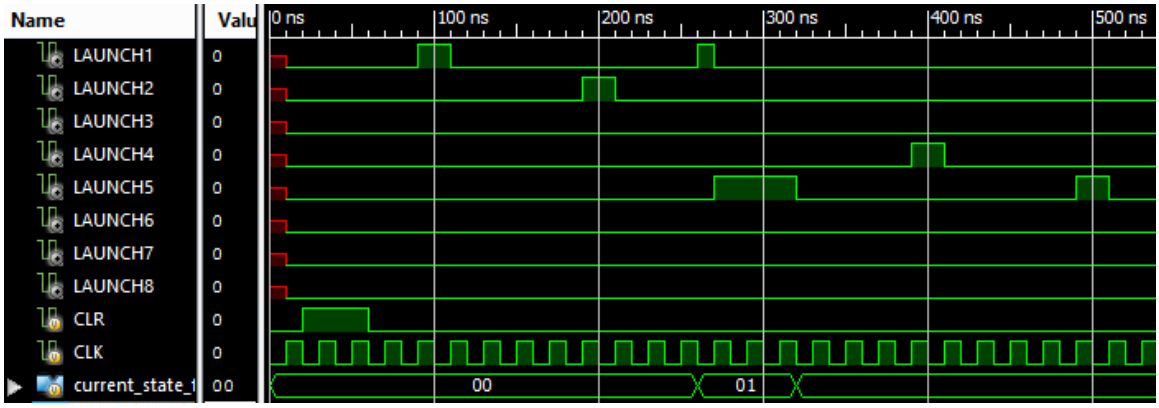


Figure 40. Sequencer operation with a fault in the MSB of the state variable.

Several other test cases were performed for the LSB and MSB of the state output to observe the effects of changing the time and duration of the error. Significant problems were seen in every case where the error was inserted during a positive clock edge. To maintain consistency with the following tests, only the first test case presented here is used for comparisons.

4. State Output Error—LSB

An error inserted into the LSB of the state output caused only a brief delay in operation when inserted at the 260 ns point as in previous tests. During this time, shifting that particular bit simply forced the machine back into the wait state for a short time. However, at other times, the effect is more pronounced. For this case, the error is inserted at 200 ns and causes a jump from the activation of P-POD#2 to the launch of P-

POD#1. On an actual launch vehicle, this may cause a failure to activate on P-POD#2's NEA if the signal was not inserted for sufficient time.

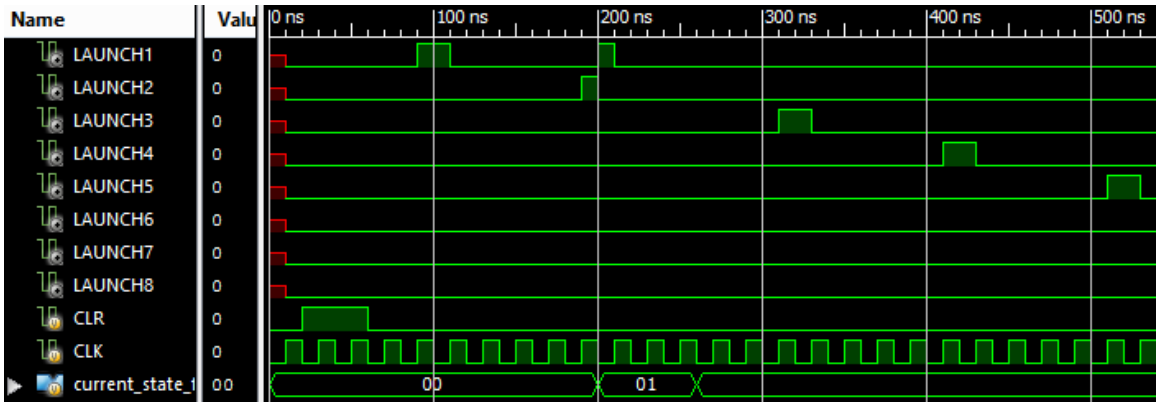


Figure 41. Sequencer operations with a fault in the LSB of the state variable.

C. MANUAL TMR CONFIGURATION AND PERFORMANCE

The sequencer design implemented here requires a fairly small footprint on the FPGA. To ensure the highest levels of reliability, a distributed TMR scheme was chosen, with a unique voter set for each of the triplicated sequencers but a global clock and reset line. In this version of the design, each of the three modules has its own set of NEA control outputs. These are combined via the hardware voter discussed in Chapter III in an actual hardware configuration. A block diagram of this configuration is shown in Figure 42.

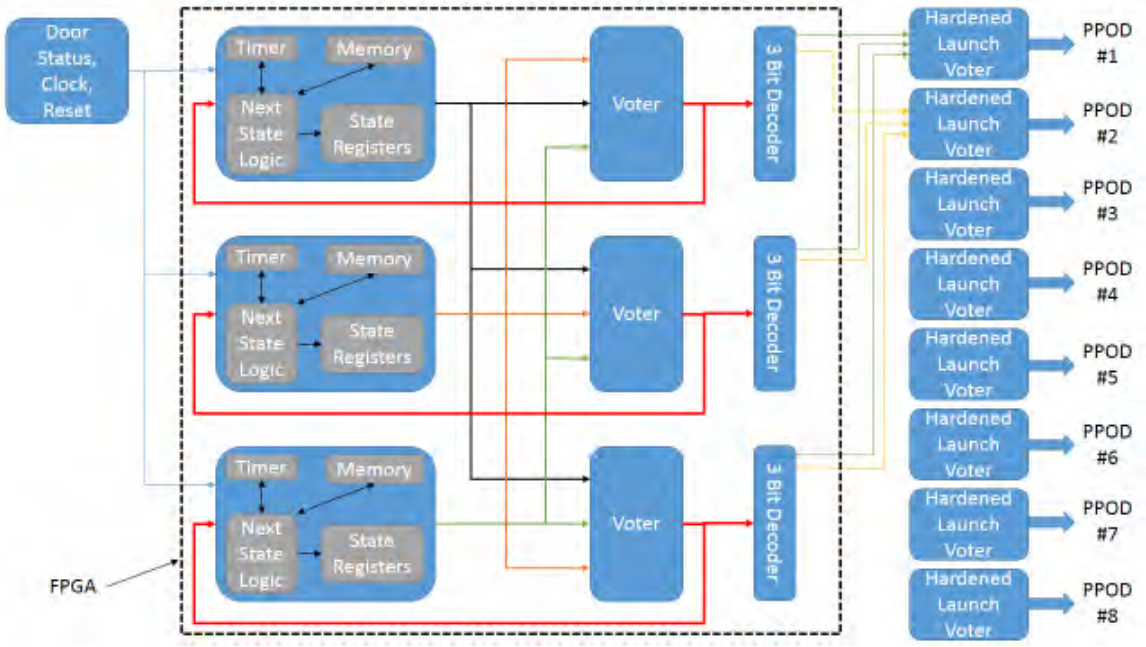


Figure 42. Block diagram of a DTMR sequencer with hardened launch voters.

For the software simulation version discussed here, the outputs of the three voters are combined with a simple software voter before being combined by the launch decoder. A diagram of this modified version can be seen in Figure 43. This was done for several reasons. Primarily, the purpose of this simulation was not to test the function of the hardware voter design. Adding the additional modules to simulate the voters does not prove their functionality but only demonstrates the functionality of the simulation. Secondly, adding these additional modules complicates the final design and would result in an inaccurate final determination of FPGA design overhead. The launch decoder for this design is simple, and testing of this device was not planned per the developed test plan. An SEU that occurs in this portion of the logic is negated by the hardened launch voters in the real-world design.

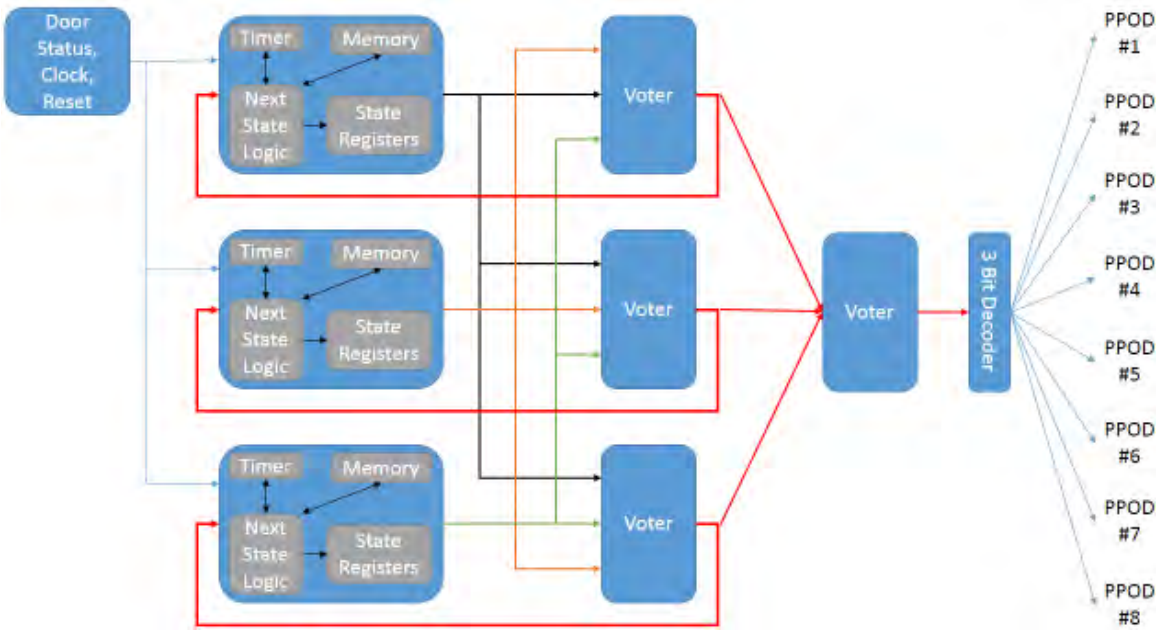


Figure 43. Block diagram of a DTMR sequencer with a single decoder and output voter.

The core of the TMR concept, the actual voter modules themselves, were created next using the basic logic gate constructs of Verilog to keep the final result as simple as possible. A one-bit voter that satisfied the Boolean equation

$$A \cdot B + A \cdot C + B \cdot C = Z \tag{1}$$

was created. This logic provides the majority vote of the *A*, *B*, and *C* inputs into the *Z* output. Four of these one-bit modules were combined into a state voter module.

An additional module was created with a selectable fault insertion to test an SEU internal to a TMR module. The error here is a simple inversion of a single bit of the voter output. An error on the voter input or certain portions of the first layer of gates is absorbed by the voter function and not seen outside the circuit. The chosen error is the worst case situation for a voter SEU. Using the fault module developed in the previous section, we applied the global TMR principles.

As in the previous case, an initial simulation was performed to establish a baseline case with no faults inserted. This case is shown in Figure 44. The operation and performance of the TMR version is identical to the basic non-TMR sequencer from the

previous section, with the launch sequence executing exactly as before. This demonstrates that, at least behaviorally, the addition of the TMR logic does not result in any changes to the operation of the design.

It should be noted that this simulation does not take into account the gate delays generated by the additional logic. When tracing signal paths through the TMR design, only two additional gates have been added in the state output path. Given the high performance of most FPGA devices and the low clock speeds at which the sequencer operates, this delay is miniscule and is disregarded for this testing.

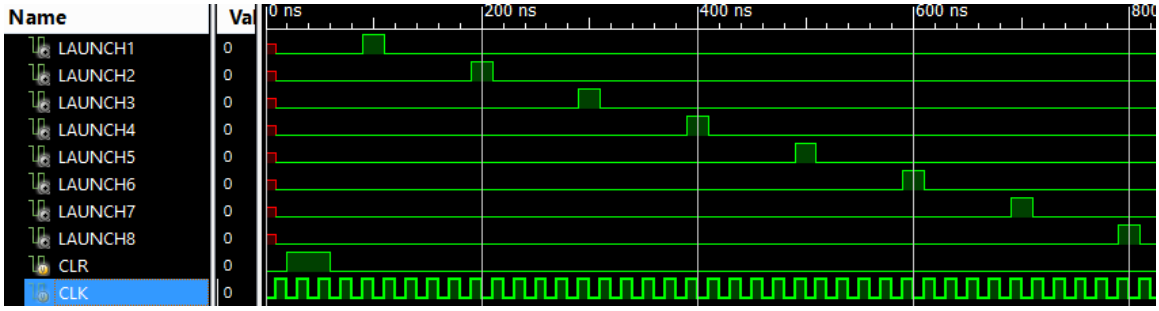


Figure 44. Manual TMR sequencer operation with no inserted faults.

1. Timeout Fault

Using the same fault insertion times as the previous case, we inserted a fault into the timeout line in one of the three timer modules. The fault signal can be seen being inserted and removed in Figure 45. There is no impact from this fault on the output of the sequencer. In this case the affected module would have delayed the launch signal, but the other two modules indicated the correct launch signal. The voter modules used the majority vote and updated all three modules with the correct next state.

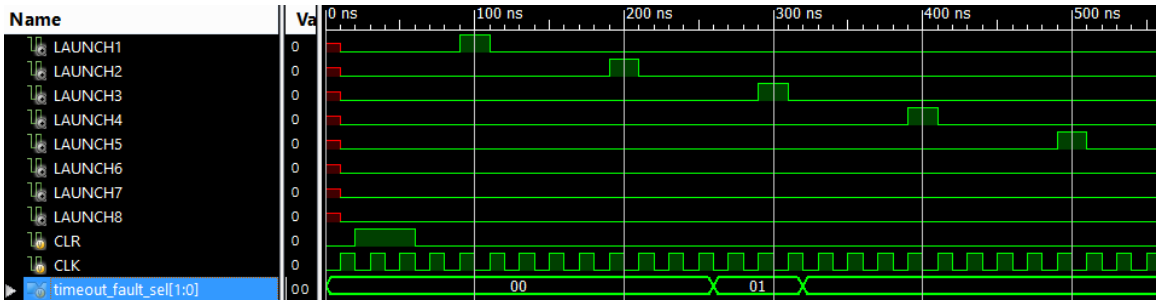


Figure 45. Manual TMR sequencer operation with a fault in the timeout signal.

2. P-POD Select Fault

A P-POD select fault was inserted at the 260 ns point and removed 60 ns later, exactly as in the single sequencer test performed earlier. As shown in the following waveform, the single fault has no impact on the overall output of the design, with the voter modules correcting the machine to the correct state before the error propagates to the output.

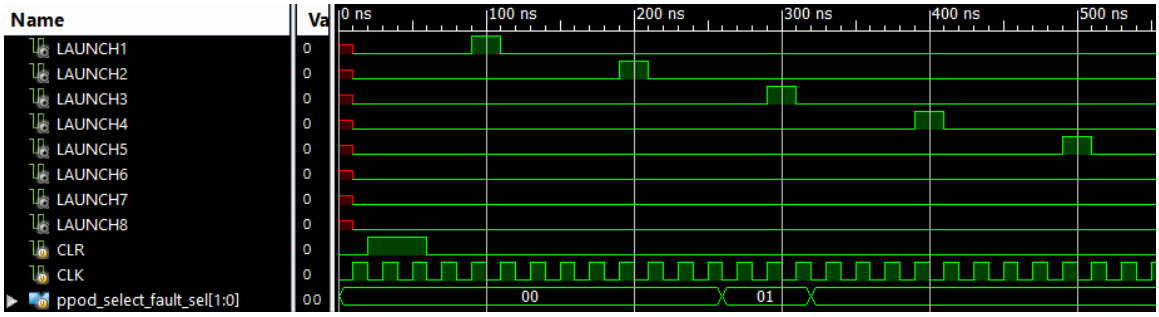


Figure 46. Manual TMR sequencer operation with a fault in the P-POD select signal.

3. State Output Error—MSB

The state output errors were evaluated next, with an error inserted in the MSB of the state output bus. In the single sequencer case, this type of error had the most significant effect on the overall operation. The manual TMR can correct this type of error as well, leaving the overall output unaffected.

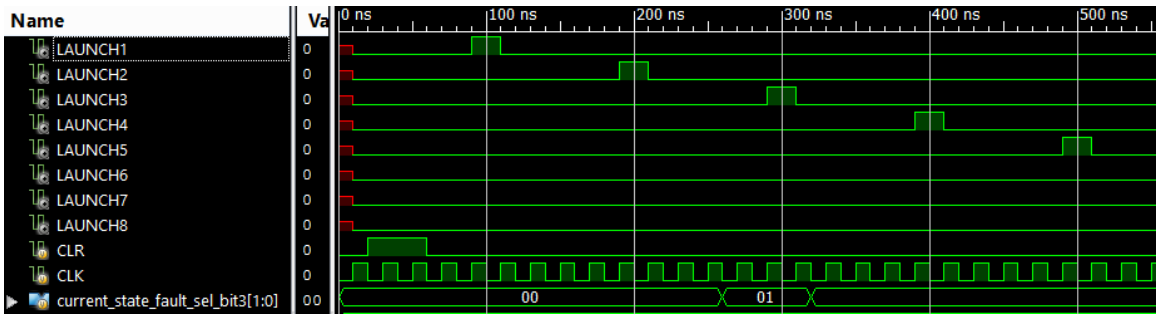


Figure 47. Manual TMR sequencer operation with a fault in the MSB of the state variable.

4. State Output Error—LSB

An error in the LSB of the state output bus of one module also has no effect on the output of the TMR version of the sequencer, as shown in Figure 48.

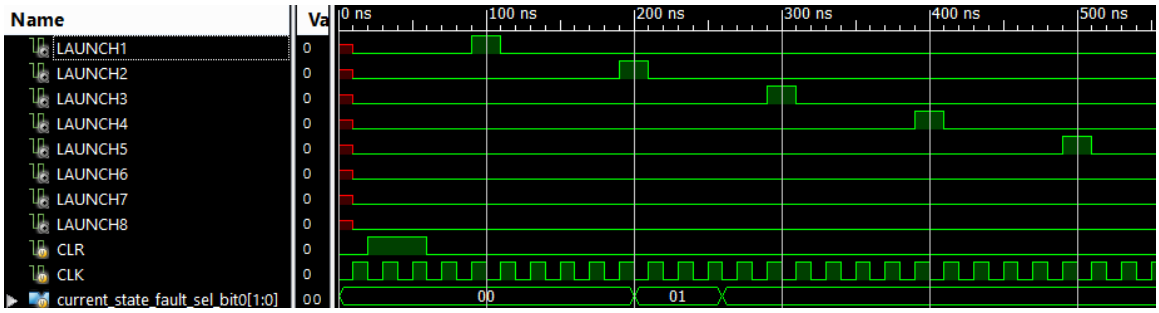


Figure 48. Manual TMR sequencer operation with a fault in the LSB of the state variable.

5. Voter Module Fault

In this particular simulation, a fault was inserted into one of the voter modules. The fault inverts one of the state output bits from the voter module itself. The erroneous signal is inserted at the 200 ns point and removed 100 ns later. A longer time was selected to allow the signal to fully propagate back to the associated module. As shown in Figure 49, the error has no impact on the overall performance. The error signal only propagates to the final launch voter, where it is filtered out. In a real hardware implementation of this design, the final launch voter would be replaced by the hardened launch voters, which would serve the same function.

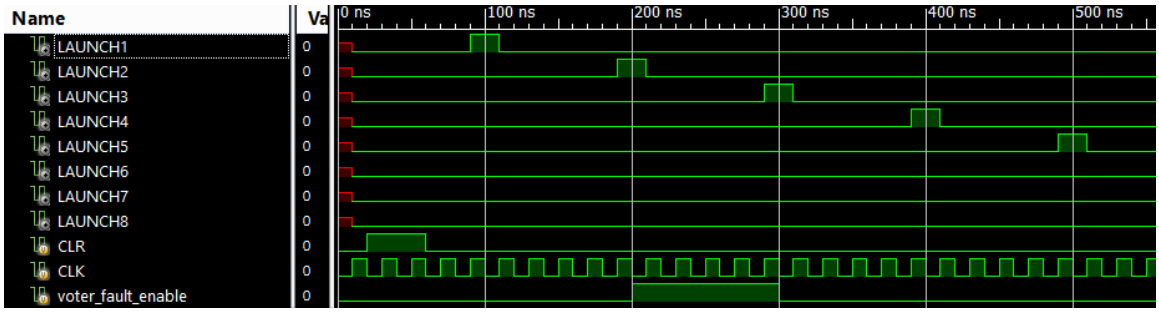


Figure 49. Manual TMR sequencer operation with a voter module fault.

D. SOFTWARE TMR CONFIGURATION AND PERFORMANCE

For a design such as this sequencer, it is possible for a single engineer to create a working TMR or other fault tolerant scheme to provide additional reliability for a system. As the design becomes more complex, adding these features becomes more and more complex and time consuming. The FPGA industry realized this, and several of the major manufactures have developed software tools to automate the process. In theory, these tools should be able to take a single design and produce a triplicated design that performs to the same level as an engineer making the design manually but in significantly less time.

The three products considered were Xilinx's TMRTool[9], Mentor Graphics' Precision[10] Synthesis, and Synopsys's Synplify[11] Premier. Unlike the main development environments, the manufacturers do not generally offer their TMR software packages as free downloads for purely academic uses. Some of the fault tolerant algorithms and information is used by U.S. space programs as well as U.S. nuclear programs, and this the technology is restricted by International Traffic in Arms Regulations (ITAR) laws and Export Administration Regulation (EAR) controls. Due to these restrictions, only the Synopsys software, Synplify Premier, was available for use in this design.

[9] TMRTool® is a registered trademark of Xilinx, Inc.

[10] Precision® is a registered trademark of Mentor Graphics Corporation

[11] Synplify® is a registered trademark of Synopsys, Inc.

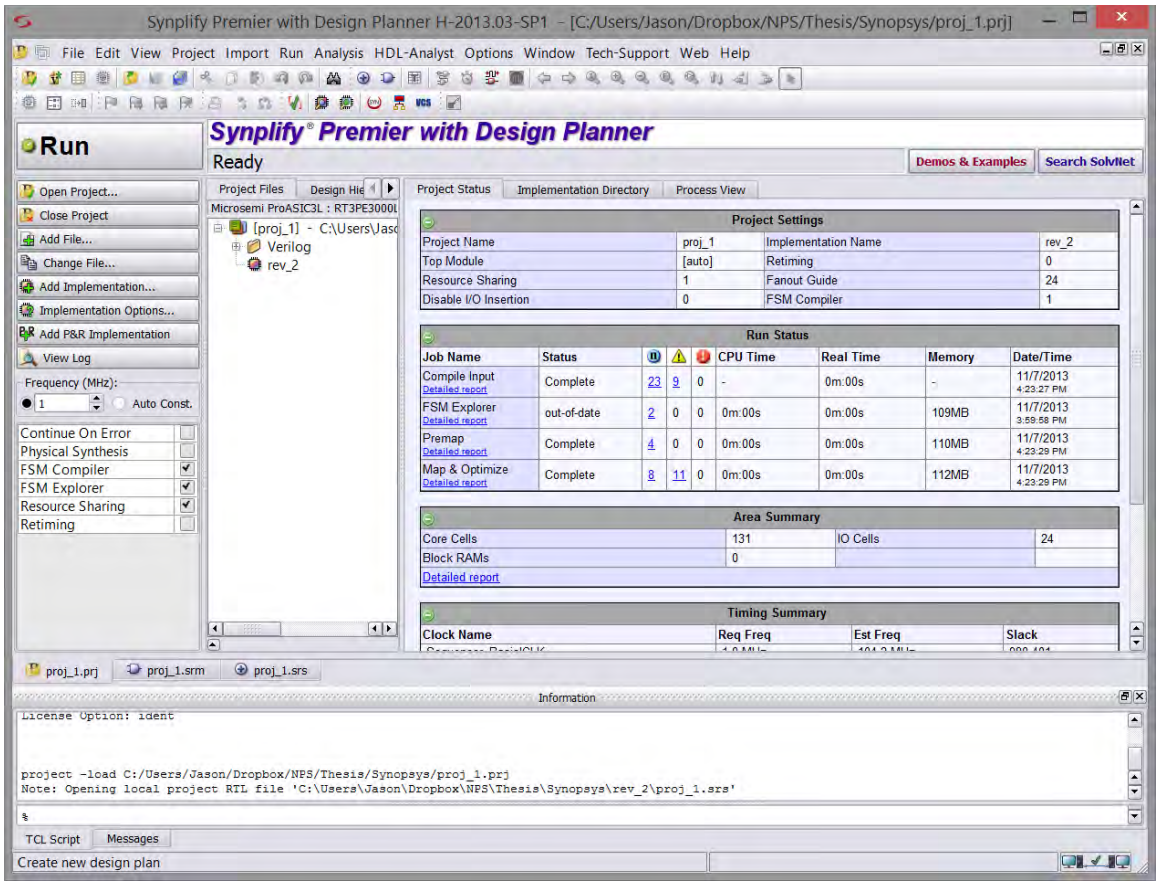


Figure 50. Synopsys Synplify Premier with Design Planner main interface screen example.

Transferring the basic sequencer design to the new development environment was done by exporting the source Verilog files from the Xilinx ISE software. These files were then imported into the Synplify software as a new project. The Synplify software allows implementation of TMR features using an attribute assigned in the Verilog code. This attribute, "syn_radhardlevel" can be assigned to an entire architecture, module, or down to the single register level [49]. There are three different rad-hard levels allowed by this implementation, combinatorial cell (CC), TMR, and TMR with CC. Here the term CC is used to describe a technique specific to Microsemi's FPGAs where additional FFs are formed using unused combinatorial logic on-chip. Microsemi's testing has indicated that FFs created using this method provide additional resistance to SEUs [50].

To maintain consistency with the manual TMR version that has already been created and to allow an "apples to apples" comparison in regard to FPGA resources and timing, the standard TMR method was used.

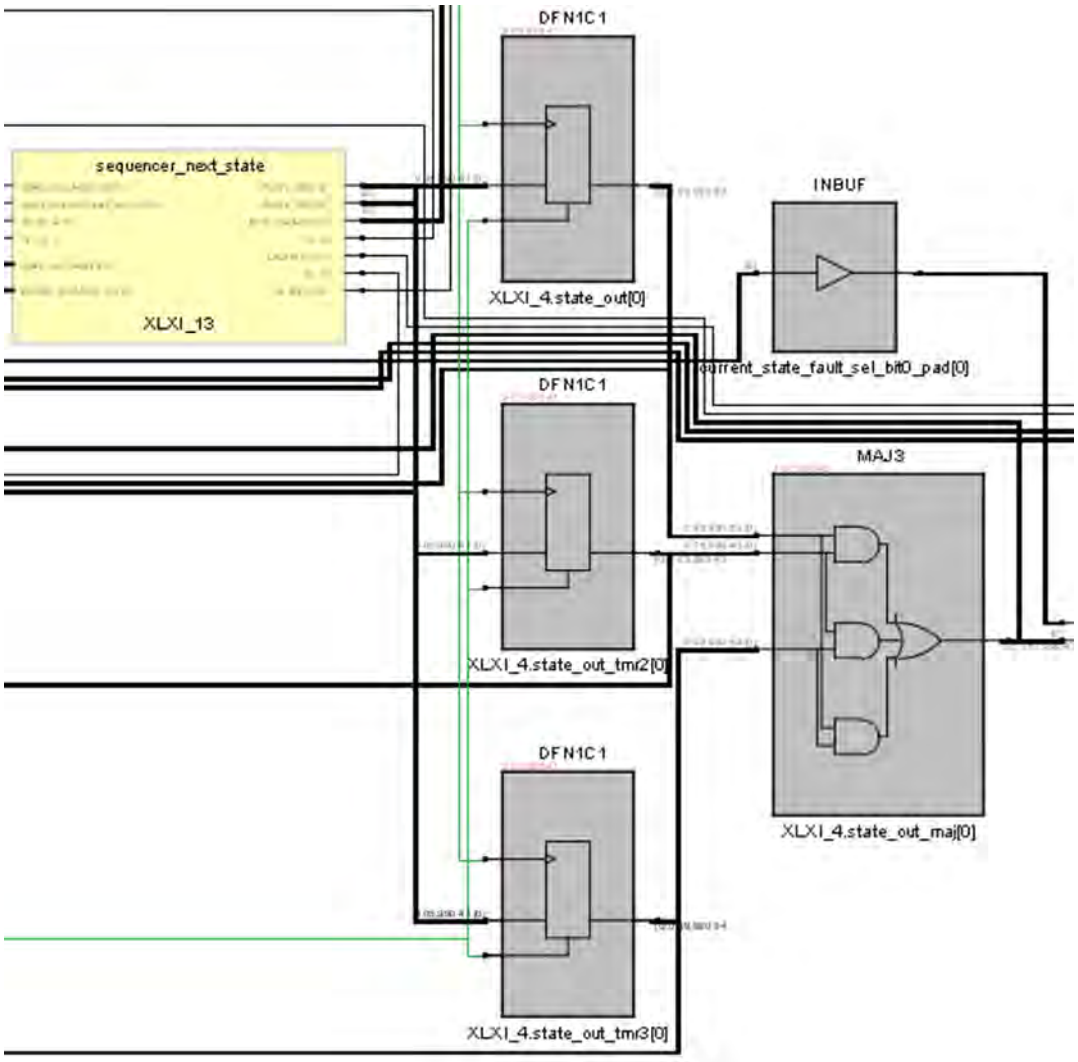


Figure 51. Synplify's TMR feature applied to an output, showing software generated FF's and majority voter.

1. TMR on State out Registers

For the initial test of the TMR features, the software was configured to perform its TMR function on the output of the state registers only. This setting resulted in the design shown in Figure 52. While this result is a useable and functional design, it does not

provide the required level of protection from SEUs. Any errors that occur in the state outputs are properly corrected by this implementation. Any faults that occurred in the next state logic, timer, or memory module propagate to the output.

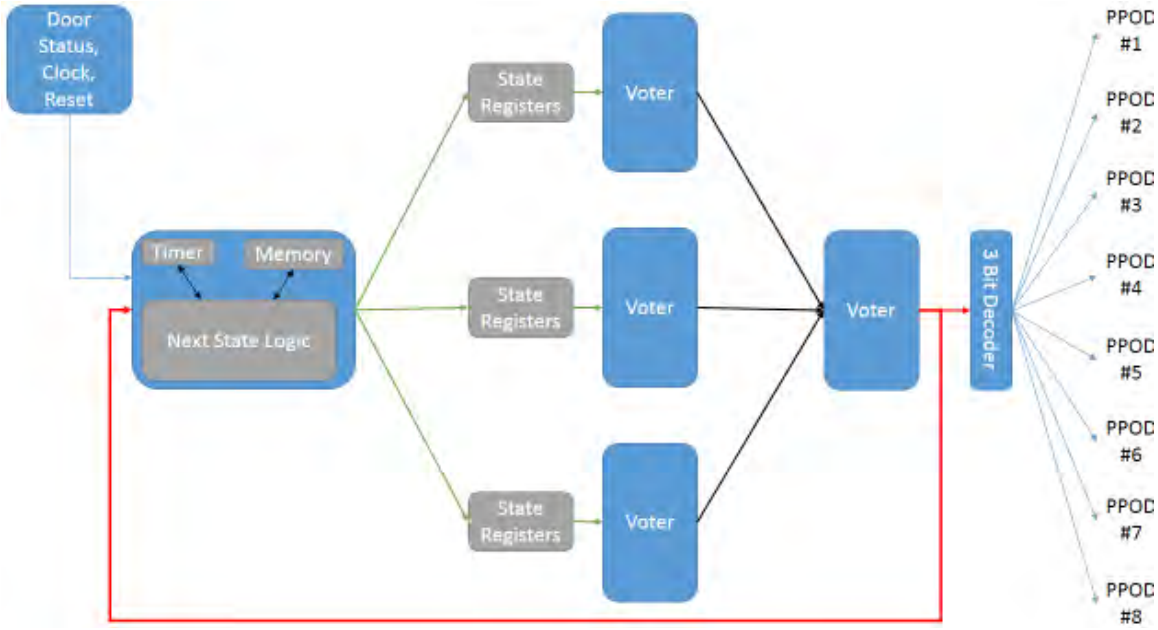


Figure 52. Design results of automatic TMR applied to state output bus only.

2. TMR on Top Level Sequencer Module

To try and provide an equal level of TMR protection with the manual TMR discussed previously, a second attempt was made using the Synplify software. In this case, the TMR option was removed from the state output registers and instead applied to the overall sequencer module. The result was closer to the desired result but still not complete. The software applied a TMR solution to the state registers and the registers in the memory and timer module but not the overall design. As shown in Figure 53, this clearly provides additional fault tolerance over the first design, but the increase in complexity introduces additional problems. The added gate count from the additional voters introduces more locations for SEUs to occur. This solution also leaves the next-state logic as a single point-of-failure for the design.

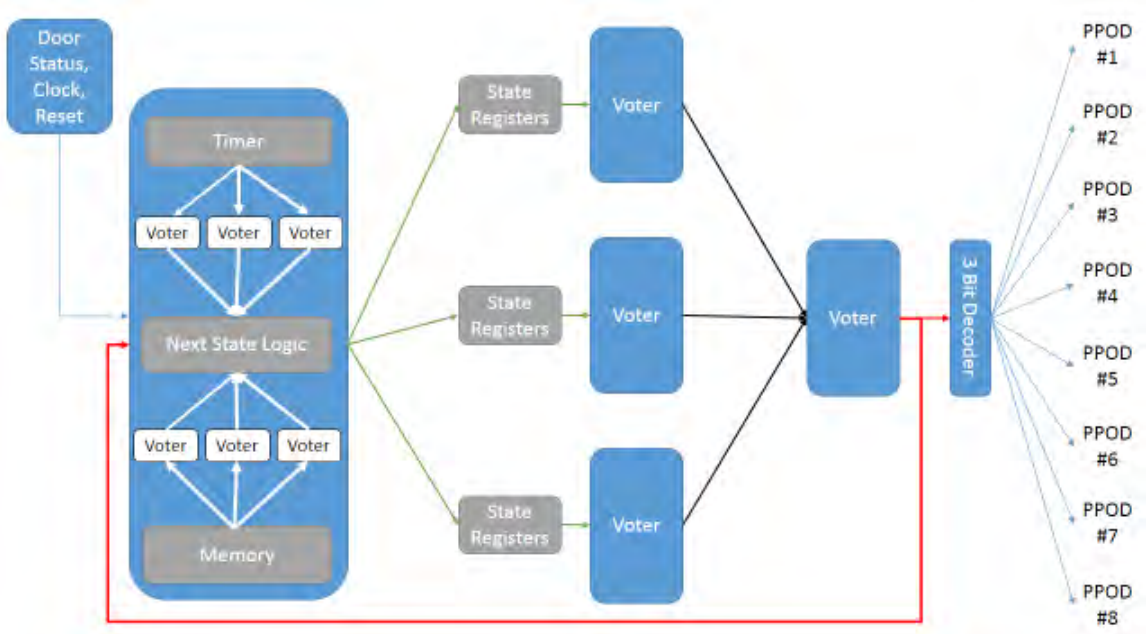


Figure 53. Design result of TMR applied to top level sequencer module.

Several other attempts were made to apply the TMR attribute to different modules and signals in the sequencer to try and obtain a result similar to the manual TMR method. In each case, simple combinational logic was ignored by the software synthesis. There seems to be no method to force triplication of the combinational logic blocks.

The Synopsis synthesis software does not provide any built-in simulation software. To perform a fault-tolerance and operational comparison between the software TMR version generated here and the manual version, an extra step was required. The generated design was exported in schematic form from the Synplify package. This schematic was than manually imported as a new design into the Xilinx ISE package and from there the previously used ISim simulations were performed.

The more complex design generated by this synthesis software raised an additional challenge for testing. Specifically, a decision had to be made as to where to insert the test signals. Using the sequencer memory module as an example, the TMR software triplicated the output registers and added a voter circuit, as shown in Figure 54.

If an SEU occurs in the register itself, this TMR method corrects the problem. If the SEU occurs in the voter output module or the input to the memory module, then the error propagates throughout the design.

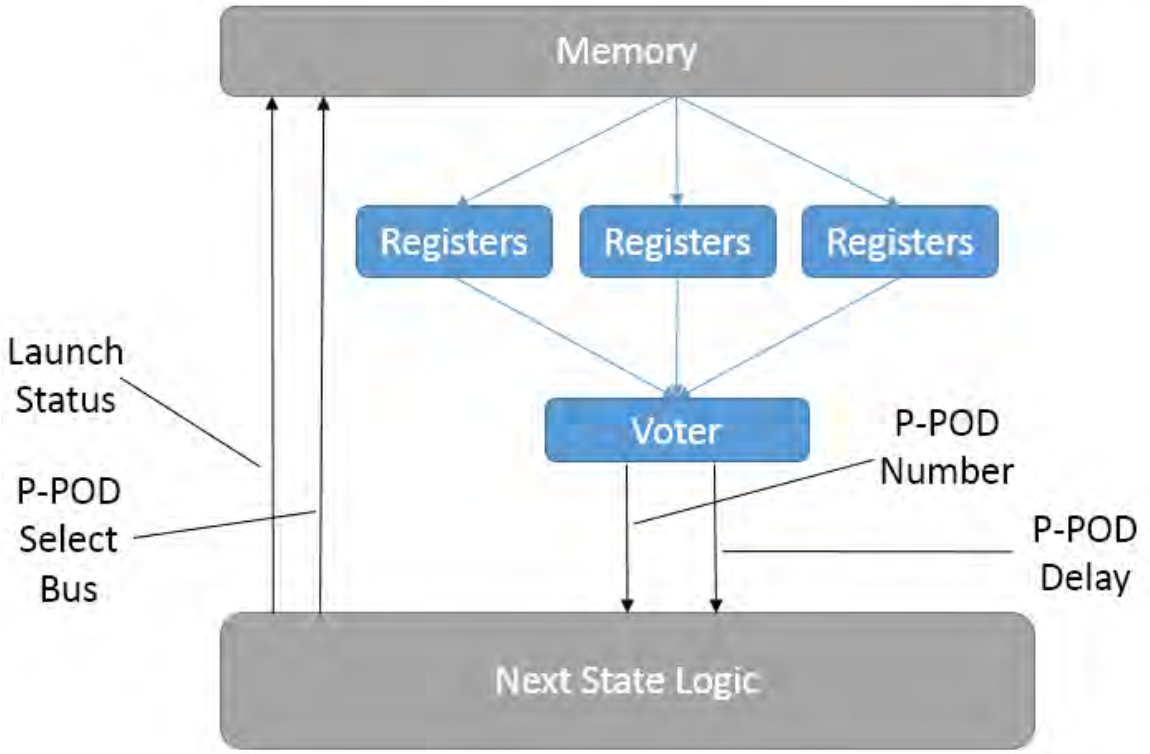


Figure 54. Detailed view of software TMR on the memory and next state logic interface.

To verify this issue and observe its effects on the performance of the software TMR sequencer design, the fault modules were added in different locations for the memory and timer modules and are discussed in their respective sections. The software TMR's version of the state registers and voters was very similar to the manual TMR version, and it was possible to insert the state output faults in the same locations to achieve the most accurate comparison. A first simulation was performed with no inserted faults, and the output is identical to the single and manual TMR cases, as seen in Figure 55.

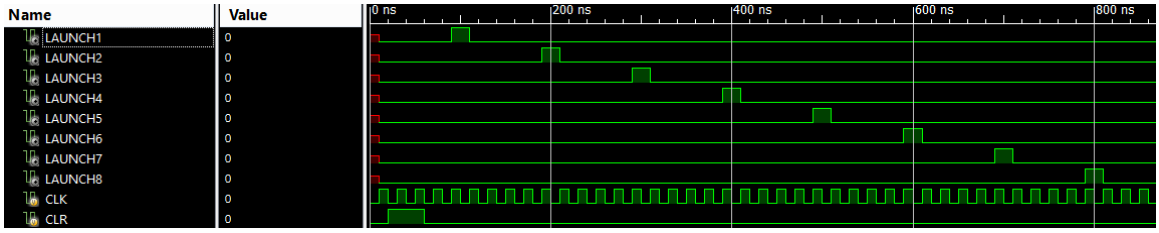


Figure 55. Software TMR simulation with no inserted faults.

3. Timeout Fault

For the timeout fault case, the fault was inserted into one of the triplicated timeout registers from a single timeout module as seen in Figure 56. This TMR configuration as delivered by the Synopsys software would probably handle any errors generated in the timer modules. A simulation was performed with several times and durations of error inserted, and in each case the circuit corrected the error. A final comparison run was performed using the same time from the manual TMR case for comparison, as shown in Figure 57.

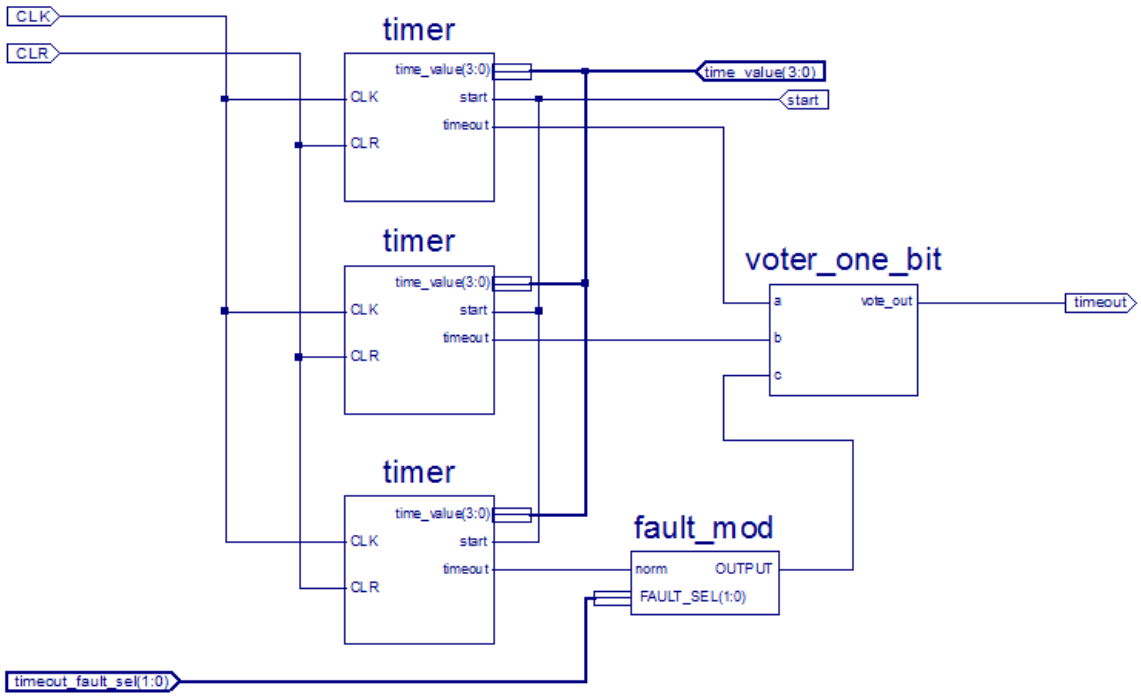


Figure 56. Software TMR timer fault insertion schematic showing fault location.

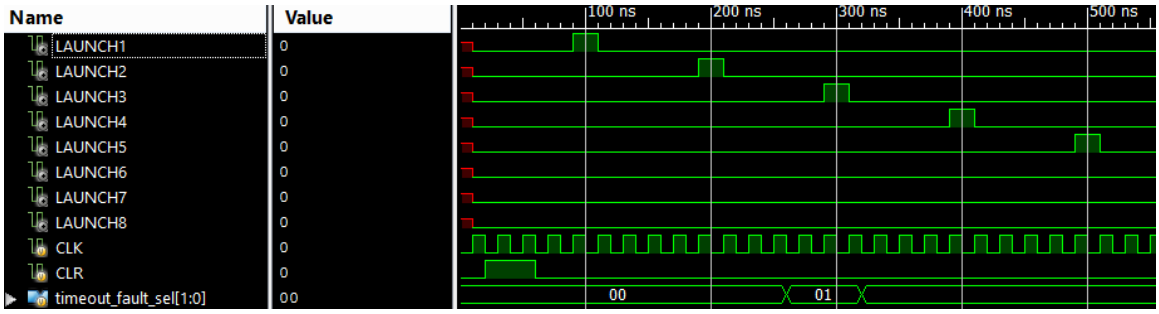


Figure 57. Software TMR sequencer operation with a fault in the timeout signal.

4. P-POD Select Fault

For the P-POD select fault, the error was inserted into the design between the next state logic and the memory module. As the software TMR solution only applied the additional registers and voting logic to the module output, no fault protection was provided. As shown in Figure 58, the error propagated through all three output modules. Since all three modules produced the same error, the voter was not able to resolve the issue. Comparing this result to previous cases, we see that the output is the same as the single, unprotected module seen in Figure 39.

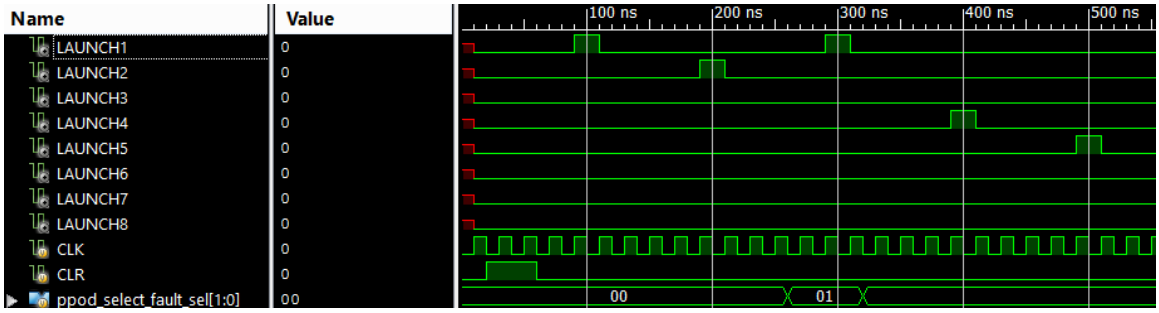


Figure 58. Software TMR sequencer operation with a fault in the P-POD select signal.

5. State Out Errors

The software TMR design and manual TMR designs are nearly identical in their output with regard to how the state registers and state information is handled. A number of tests for faults on the MSB and LSB were performed at various insertion times and durations. These faults had no impact on the operation of the overall sequencer, as expected in this TMR configuration. A combined case, with errors in both the MSB and

LSB is shown in Figure 59. A fault in the MSB occurs first at the 260 ns point, and a fault on the LSB occurs at 460 ns. Neither of these errors affects the output of the sequencer.

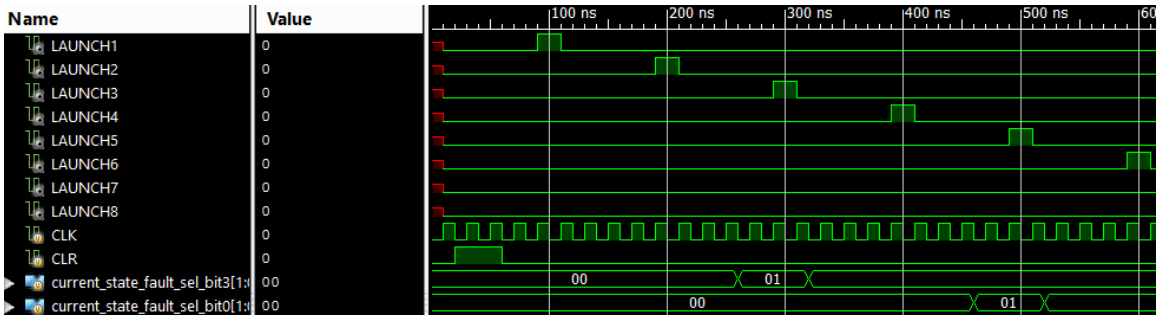


Figure 59. Software TMR sequencer operation with faults in the state output MSB and LSB.

6. Voter Module Fault

The potential problem caused by an SEU in one of the voter modules is significantly increased with the software TMR solution. This design contains three times the number of voter modules as the manual TMR version, and faults are more likely due to the increased physical area used by voters on the FPGA. Several experiments with voter faults were explored.

a. Timer Voter Faults

Faults in the timer module voters had results very similar to timer faults in the non-TMR sequencer. This is to be expected, as the software TMR applied only protection to the timeout signal.

b. Memory Voter Faults

The software TMR solution protects the output registers from the memory module, specifically the three bit bus that designates the "current" P-POD and the four bit bus containing the required launch delay. Voter errors in these modules created significant problems with operation that were not evaluated in the non-TMR version. Faults inserted here can cause the incorrect P-POD to launch or improper launch delays.

c. State Voter Faults

A fault in the first layer of state voters has no impact on the overall operation of the software TMR version of the sequencer. Comparable to the manual TMR version, any error in the voter is absorbed by the output voter. Unlike the manual TMR version, the software version incorporates a single output voter. Since there is only one next state-logic module, the software TMR version must have a final voter before the signal is looped back to the input of the next-state module. In the manual TMR version, this final state voter was for simulation only and would be replaced with the hardware voter in an operational design. This introduces an additional point of failure for the software TMR version, as seen in Figure 60. Here a voter fault introduces a state error that cannot be corrected by the hardware voter output as in the manual TMR case.

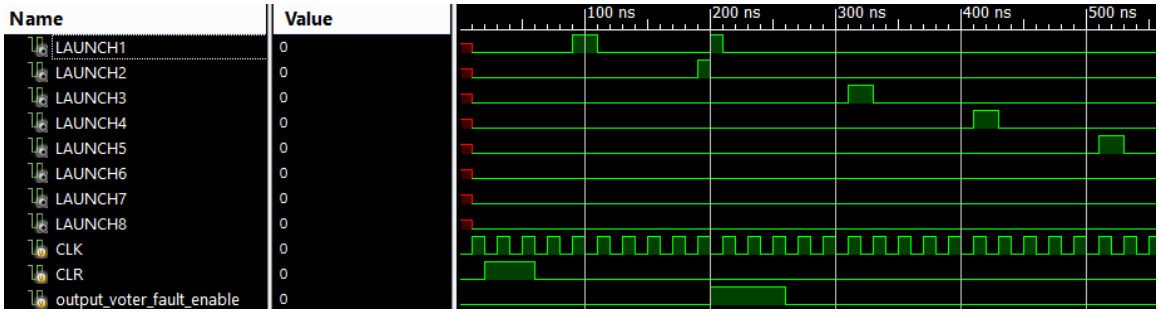


Figure 60. Software TMR sequencer operation with a fault in the second layer state voter circuit.

E. CHAPTER SUMMARY

In this chapter the design for the basic sequencer was moved from the concepts of Chapter III into a version that could be simulated in software. A brief review of available software packages was performed, and a development environment was selected for future use. The potential effects of SEUs on the FPGA were evaluated and translated into the expected types of digital logic errors that can occur. A method was then developed to simulate those errors in software, and the locations for each error were established for consistent testing.

To establish a baseline for comparison, a series of faults were simulated on a basic version of the sequencer with no fault tolerant features. The effects on the operation were evaluated for each fault condition. These tests were then repeated on a version of the sequencer with TMR protection applied by hand and again on a version with TMR protection applied via a software tool. Comparisons were made between each version and the capability of the TMR protection to provide fault-free operation for the given error conditions.

Upon review of the results in this chapter, there is no doubt that TMR can provide considerable protection from SEUs. The manual TMR case was most effective in these tests, as it was able to deliver fault-free operation in every case. The software TMR version afforded some protection but was not able to avoid error in all cases. The unprotected logic and the input signals to the protected sections were vulnerable to errors, as demonstrated in the output errors that occurred from errors in the P-POD select lines. In addition, the design produced by the software introduced some additional sources of error, which may further degrade the performance.

For this level of design complexity, manual TMR is not time consuming and provides superior results to software TMR. As complexity increases, the time required to apply manual TMR also increases, and the potential for human errors increases. While the engineering man-hours required to complete a design are important, the following chapter discusses the other hardware factors that must also be considered.

V. HARDWARE IMPLEMENTATION AND ANALYSIS

A. FPGA IMPLEMENTATION

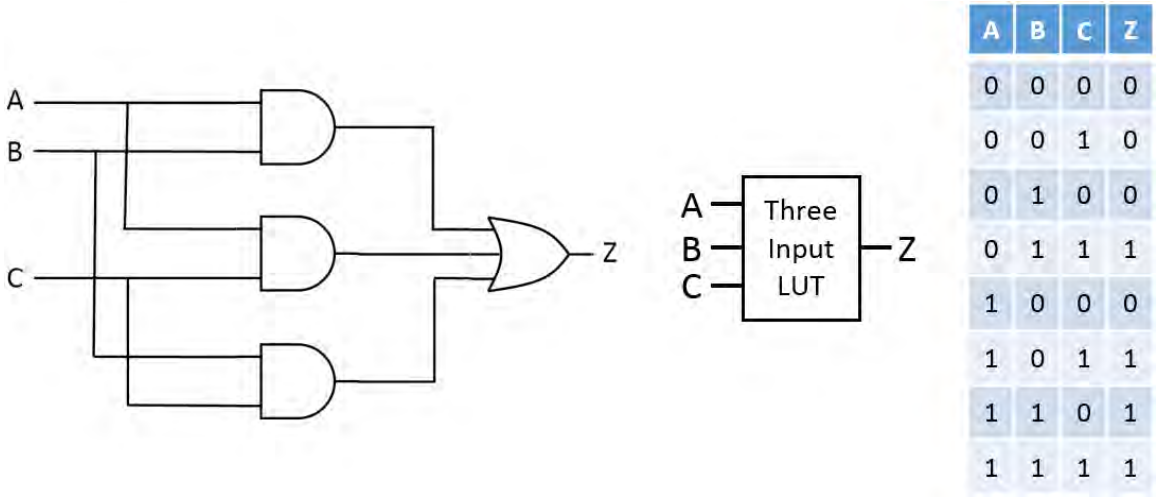
Translating the design developed in previous chapters onto a physical FPGA is a complex process. A typical FPGA design flow starts with design entry, then moves to synthesis, place and route, timing analysis/simulation, and finally device programming. A brief overview of these steps is covered here to review particular issues for the sequencer design and fault-tolerance.

1. Design Entry

Design entry is the initial step of FPGA development and was completed and tested in the previous chapter. The end result of this step is either a Verilog or VHDL design or, in this case, a combined schematic and Verilog design. The schematics used here are purely for ease of understanding and visualization for the designer. The development software converts the schematics to an equivalent Verilog file in the last stage of this step.

2. Synthesis

Synthesis is the process of converting the high-level circuit description provided by the HDL into a low-level description using gates and on-chip components, commonly called register-transfer level (RTL). High performance FPGAs generally do not use basic gates on chip but instead use a device called a look-up table (LUT). These are essentially small memory devices that are programmed with the equivalent logic table. A LUT generally has a lower gate delay and takes less FPGA resources than the equivalent gate structure, making them both faster and more space-efficient. An example for the voter circuit is shown in Figure 61.



A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Figure 61. Conversion of a majority voter logic gate design into a LUT implementation using a three-input LUT.

In order to synthesize a design for a given FPGA, the software must first know the exact internal structure of the FPGA to be used. For example, the design above uses a three-input LUT, but the larger Actel and Xilinx all use four or six input LUTs. The knowledge of the internal routing structure, clocks, registers and all other on-chip components is critical to producing a working design. This data is generally built-in to the development environment, and this is one reason the major FPGA manufacturers also produce their own development software.

A significant level of design optimization also occurs during this step, and this presents a potential problem for the fault-tolerant designs. FPGA designers are generally concerned with two major optimizations, speed and areas. The development software automatically reviews a given design, rearranges components and removes redundant or unused gates or signals. An example of the optimization of the timer module, detailing the final register transfer level (RTL) schematic, is shown in Figure 62. This optimization is critical if the designer is trying to meet certain timing goals or if the design is large and may not fit on the FPGA otherwise. Unfortunately, this optimization is detrimental to TMR designs, as it detects the redundant components and trims them from the final design. Additional command switches and software options must be set to ensure this does not occur when developing a TMR logic design.

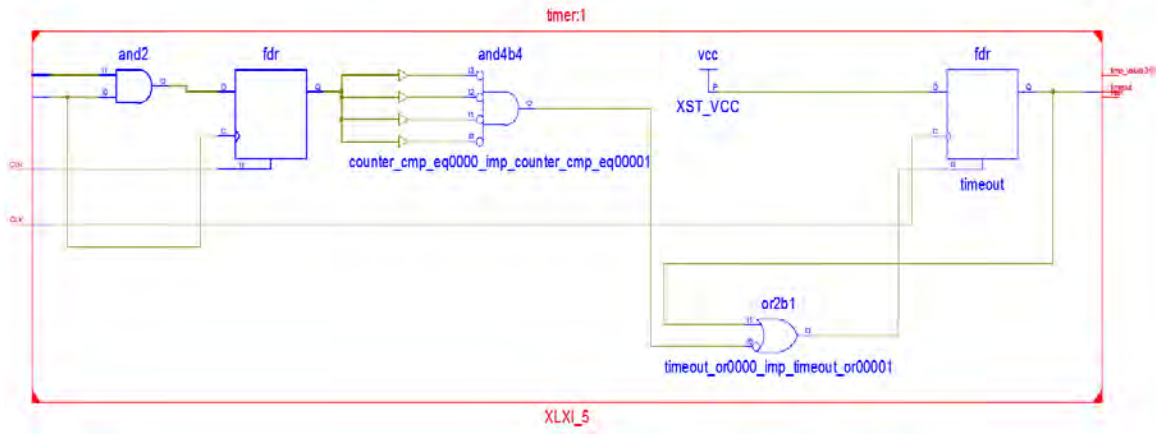


Figure 62. RTL schematic produced following synthesis of a timer module in the sequencer design.

3. Place and Route

In the place and route step, the design produced from synthesis is mapped into the physical logic blocks and I/O pads of the target FPGA. A set of selected user constraints is loaded first. This maps the I/O pins on the FPGA itself to the I/O markers on the logic design. Any internal clocks, memory devices, or other on-chip components are assigned to the design in this file. To complete the process, the optimized design is mapped onto the physical LUTs, registers, and other components of the target FPGA. The software generally performs additional optimizations during this step, placing components to improve performance by keeping signal lines as short as possible or placing them an exact distance apart from each other to minimize clock skew.

As seen in Figure 63, the final maps are very complex and difficult for humans to decipher. In this figure, each of the tiny grey blocks are unused logic blocks. The dark blue blocks are logic blocks that are being used, and the light blue lines are active signal connections. The red line in this figure is the signal trace for one bit of the P-POD advance signal, showing its connection between four other logic blocks.

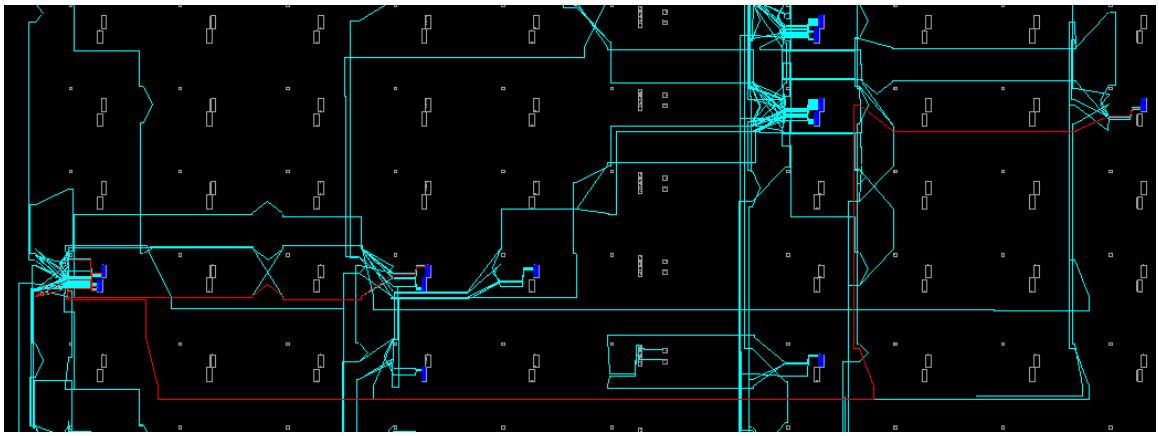


Figure 63. View of a Xilinx FPGA place and route map for the sequencer design.

B. FPGA METRICS

To compare the three sequencer designs developed here, they were run through both the synthesis and place and route steps. As before, the Xilinx ISE 14.6 was used in both steps. For consistency and future use on the available development hardware, the designs were targeted for the Xilinx Virtex-5 device, specifically, the XC5VLX50T. Several non-standard options were set in the synthesis options to prevent trimming of the redundant logic by the software as follows [2]:

- Equivalent Register Removal set to FALSE. This prevents the trimming of duplicate state registers.
- Resource Sharing set to FALSE. This prevents the synthesis software from attempting to share on-chip resources, such as adders or MUXs between modules, which would eliminate desired redundancy.

1. Synthesis Results

Following full synthesis of each of the three designs, we reviewed the synthesis report and the low-level summary results were recorded, as can be seen in Table 5.

Table 5. FPGA Resources used for three different sequencer designs

	Sequencer Design Type		
Resource	Single	Manual TMR	Software TMR
Inverters	6	6	6
Two Input LUT	7	21	23
Three Input LUT	8	27	20
Four Input LUT	14	26	13
Five Input LUT	6	26	5
Six Input LUT	21	44	19
MUX	1	10	1
Flip-Flops / Latches	14	42	28

The results here are consistent with what was observed from the block diagrams. Due to the triplication of entire modules, the manual TMR version consumes three to four times the resources of the single sequencer. The software TMR version falls between the two in terms of resources but is closer to the manual TMR version. As would be expected by the nature of TMR, adding this fault protection results in an overall design that is at least three times larger than the original.

In each case explored here, the total use of the FPGA was reported as less than one percent of the total available resources. For example, the FPGA selected has 28,800 LUTs available, and the manual TMR design only used 114 LUTs. The specific FPGA chosen was selected due to the availability of the development board for testing and is significantly larger than required for a flight hardware device. An example map showing the vast quantities of unused space on this FPGA is seen in Figure 64. The logic used by the manual TMR sequencer is represented by the miniscule light blue dots in the top third of the map. Using this FPGA does have the advantage of being pin-compatible with the rad-hard versions of the same FPGA, which reduces the testing costs for future use. In addition, the extra space provides significant room for any design changes or future improvements.

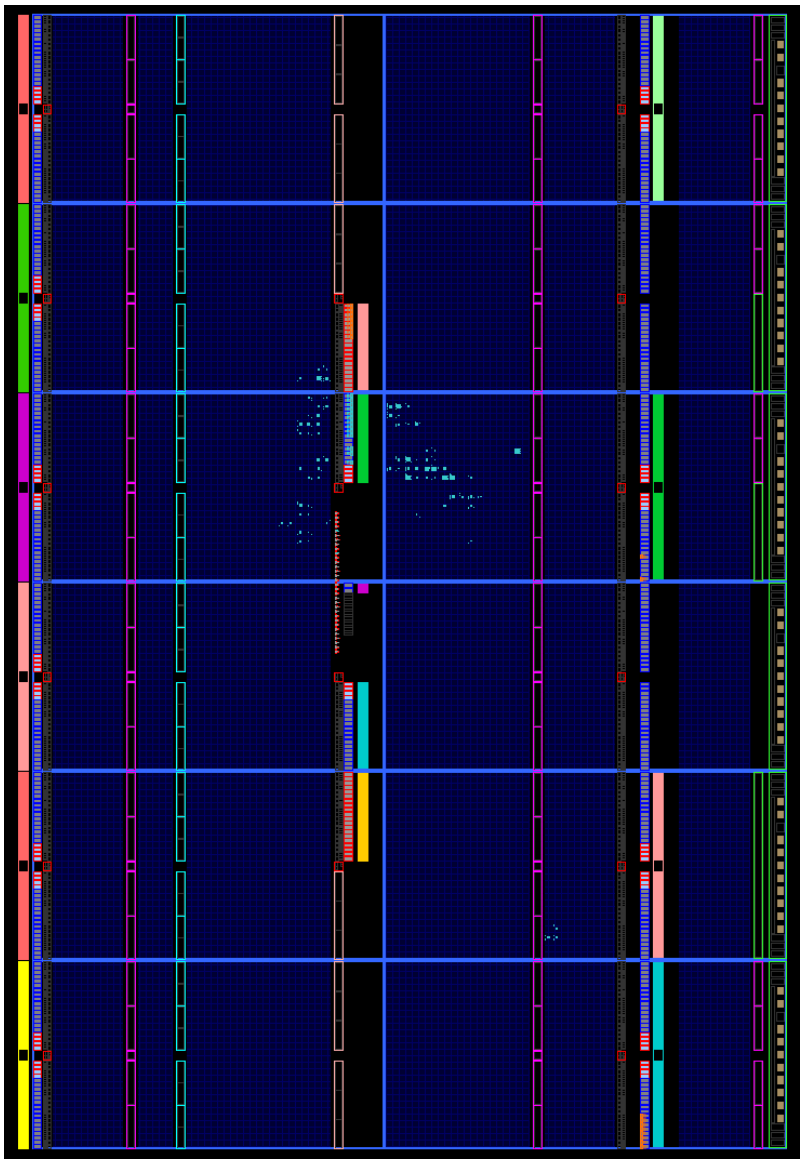


Figure 64. Manual TMR sequencer final place and route showing <1% resource usage.

2. Fault Tolerant Place and Route

Using an FGPA that is significantly larger than the design requires also offers another fault-tolerant advantage that has not been discussed. As the feature size and operating voltages of the FPGAs decreases, they become more susceptible to MBUs [51]. The software TMR design realized here cannot provide protection from MBUs in most cases. The manual TMR can provide protection for MBUs if they occur in the same module or different bit positions in multiple modules. The most robust TMR design

cannot protect against an MBU that causes a bit flip in two adjacent sequencer modules or voter circuits. Pre-planning the place and route on a design can be used in this case to spread the modules out over a wide area on the FGPA.

Using the development software, in this case, the Xilinx FPGA Editor or the PlanAhead[12] software, we can obtain a more distributed design. An example of a rough block diagram for a place and route arrangement is seen in Figure 65. Since radiation events that cause MBUs are localized to the area the incoming energy impacts the device, physically separating the modules on-chip ensures that if an MBU occurs, only a single module of the TMR scheme is impacted and the overall output remains unaffected.

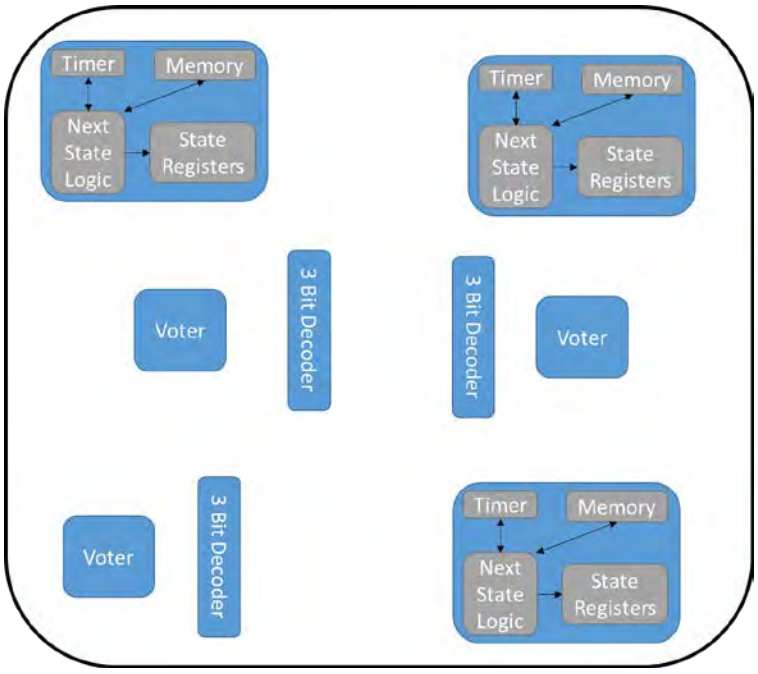


Figure 65. TMR sequencer modules distributed widely across the entire FPGA package.

This concept poses two drawbacks, both in the form of time. For a more complex design, manually re-distributing the components of the design across the FPGA is time consuming for the engineer. At the time of this writing there is no automatic software solution to evenly distribute logic across a large FPGA. The other major concern is

[12] PlanAhead ® is a registered trademark of Xilinx, Inc.

timing within the design. As each block is moved, the time for signals to move between gates is changed. In a normal arrangement, the development software places the blocks to minimize the clock skew or at least keep it within set parameters. If this routing is done manually, the clock skew may become fairly large and introduce timing errors into the design. The increase in distance between modules also increases the total FF output to input delay, increasing the allowable clock period and lowering the maximum operating frequency. For the sequencer presented here, the required clock speeds are very low, and this additional routing time does not impact the operation. In a more complex design, this could rapidly become a limiting factor.

C. TIMING ANALYSIS

Using the same settings as the previous evaluation, we generated a timing report for each design. The compiled key portions of the report are shown in Table 6. For the sequencer design presented here, timing is not a major factor in the design. The sequencer uses the clock pulses in the internal timing, so the final clock frequency is determined more by the required delay between launches and not the specifics of the FPGA technology.

Table 6. Timing comparison for fault tolerant sequencer designs.

	Sequencer Design Type		
	Single	Manual TMR	Software TMR
Minimum Period (ns)	3.132	4.654	4.952
Maximum Frequency (Mhz)	319.25	214.87	201.94
Maximum Combinational Path Delay (ns)	5.464	5.87	8.626

The current design allows four bits for time delay, so a minimum delay of one clock period and a maximum delay of 15 times the clock period is possible. Using this information the following table was generated to determine a reasonable clock frequency. In this case, even the slowest calculated frequency does not meet the design requirement for delay between launches, which requires a one second to one hour delay.

Table 7. Available P-POD delay times with various operating frequencies

	Clock Frequency				
	1 kHz	10 kHz	100 kHz	1 MHz	100 MHz
Minimum Delay (ms)	1	0.1	0.01	0.001	0.0001
Maximum Delay (ms)	15	1.5	0.15	0.015	0.0015

There are two possible solutions to alter the design to meet the initial requirements. One option is to operate at low clock speed and insert a clock divider into the timer module. This does not significantly affect the complexity of the design and provides a reasonable level of control over launch delay. Another option is to alter the designs to increase the bit-width of the timer. This change increases the size of the timer module, but it provides a very fine-grain control over the launch delay. Regardless of the method used, the timing data for the existing designs is more than sufficient to demonstrate operation at clock speeds well in excess of what will be used by an operational design.

D. TEST HARDWARE IMPLEMENTATION

1. Board Selection

The final step in testing the sequencer was loading the design onto a hardware prototyping board for testing. The key features desired were:

- An FPGA that is either already rad-tolerant or at least compatible to a rad-tolerant version. This ensures the design will be portable to a rad tolerant device for future use. Development boards with actual radiation tolerant FPGAs are considerably more expensive and unnecessary for the level of testing being performed here.
- Sufficient indications and input device onboard to allow testing without significant additional prototyping boards and components. Specifically, the board needed a minimum of eight switch inputs for the door switch inputs, two or three momentary-contact switches to act as clocks or error signals, and eight LEDs for indication of launch output signals.

Three devices were considered, the NPS ProASIC3 Test Board [2], and two general prototyping and testing boards, one from Xilinx and one from Actel.

The NPS ProASIC3 board, as the planned board for future testing with the SAD V3, was considered the ideal choice for testing, but it lacks a simple means of adding

additional input devices for design testing. To use this board for verification, an additional prototyping board needs to be manufactured with the appropriate inputs, and additional testing and troubleshooting of this new, one-time use hardware is required.

The second choice was the Microsemi ARM[13] Cortex[14] ProASIC[15] 3L M1A3PL-DEV-KIT development kit, shown in Figure 66. Similar to the NPS ProASIC3 board, this evaluation board uses the Actel M1A3PL1000 FPGA. This FPGA is compatible with the Radiation Tolerant (RT) ProASIC3 FGPA, which includes both a high level of radiation resistance, as well as built-in TMR features in the I/O banks, some FFs, and the clock network [52]. This development board includes eight dual-inline package (DIP) switches, eight LEDs and general I/OI connections that can be used to insert faults with only minor hardware changes.

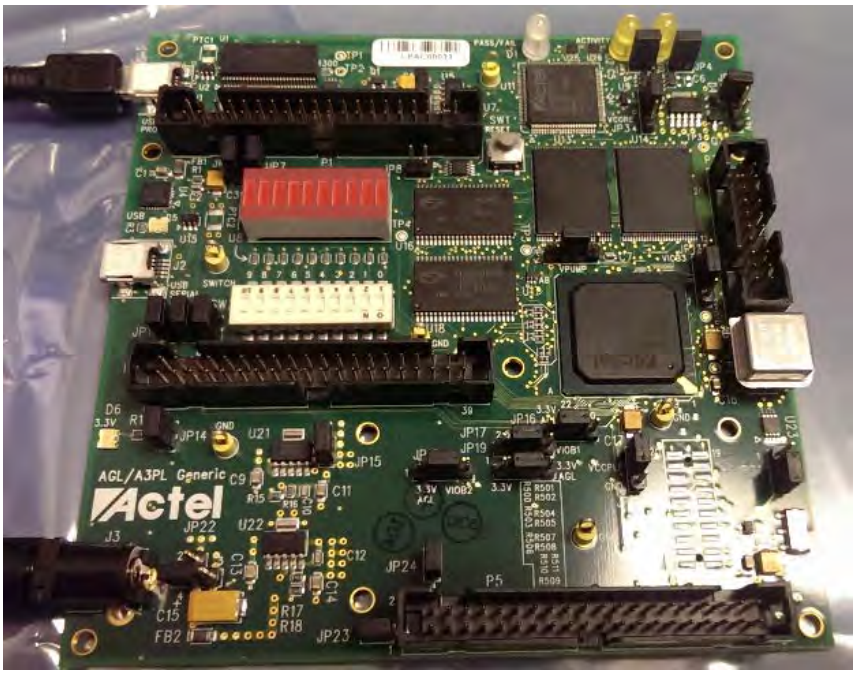


Figure 66. Microsemi ARM Cortex-M1-Enabled ProASIC3L Development Kit.

[13] ARM ® is a registered trademark of ARM Holdings

[14] Cortex is a trademark of ARM Holdings

[15] ProASIC is a registered trademark of Microsemi Corporation

The final kit considered was a Digilent Genesys[16] development kit, which employed the Xilinx Virtex-5 LX50T FPGA. Xilinx offers a radiation hardened version of this FPGA, the Virtex-5QV. This board is much more complex but offers the widest array of onboard I/O devices. Similar to the Microsemi board, a bank of eight LEDs and eight switches are offered [53]. This board also includes several momentary-contact pushbutton switches and allows for immediate use for testing with no hardware modifications necessary. The Genesys development board can be seen in Figure 67.

Figure 67. Digilent Genesys development board with Xilinx Virtex-5 FPGA.

The Digilent Genesys board was chosen for testing on this design. The criteria used for selection hinged on two factors: one, the ability to use the board immediately, with no additional hardware testing required; and two, compatibility with the

[16] Genesys is a trademark of Digilent, Inc.

development environment in use. By using a Xilinx-based product, the designs can be quickly compiled and programed using the Xilinx ISE that has been used for the majority of the prior design work. While it is possible to port the Verilog design to another environment and then program the Microsemi devices, this adds extra steps to each iteration. Using a Xilinx device also allows more accurate comparisons to the data collected previously in this thesis.

2. Design Modification for Hardware Testing

To adapt the sequencer to the physical hardware for testing, the first task was to map the I/O devices on the hardware to the I/O pads in the design files. The switches were mapped to the P-POD door status input switches. The launch signals that activate the NEAs were mapped to the on-board LEDs. The Xilinx Virtex-5 FPGA offers a wide array of internal clocking devices and multiple clock domains. This design is very flexible and powerful but overly complex for this small sequencer. The Genesys board offers a simple external 100 MHz clock pin, which is ideal for this implementation. A 32-bit counter was added to the top level of the design to act as a selectable frequency divider. For testing, the LED that is normally assigned to the launch signal for P-POD #8 was instead assigned to the clock to provide a "heartbeat" to ensure the design was operating. This allows a very precise method to step through the operation and allows time to insert various faults manually and observe the results. The on-board RESET button was assigned to the CLR signal to reset the design back to the starting condition at any time. The various error signals were mapped to the remaining momentary contact pushbuttons.

3. Hardware Testing

Basic operational testing was performed with the single sequencer design. Initial adjustments were made with the frequency divider to achieve a clock rate of about 1.5 Hz using bit 26 of the divider. Once loaded, the design properly stepped through the programmed sequence. An example, showing the P-POD launch signal at LD4

(representing P-POD #5) and the clock signal on LD7 can be seen in Figure 68. Faults inserted with the mapped pushbuttons demonstrated faults in the sequence similar to those shown in simulation.

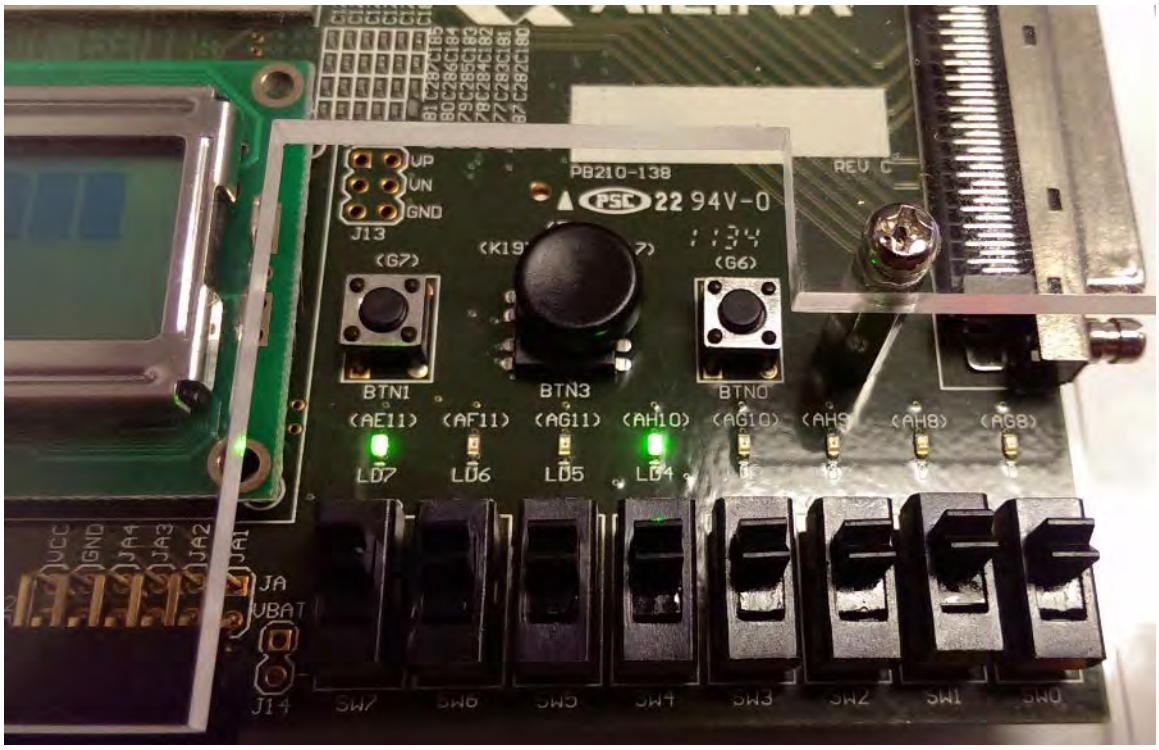


Figure 68. A demonstration of the sequencer operating on the Genesys test board.

Final testing with the manual TMR sequencer demonstrated results similar to those shown by the simulations in the previous chapter. Insertion of faults via the push buttons had no impact on the programmed sequence, indicating the fault tolerant design was working properly.

With the available hardware test devices, it is difficult to perform complete hardware level testing. A more complete test board that included connections for an external digital logic analyzer is necessary. Use of one of these devices allows capture of the actual logic waveforms inside the design while it is running in the FPGA and provides a direct comparison for the design simulations performed in the previous chapter. Without those test connections and the logic analyzer equipment, the possible hardware testing is somewhat restricted.

E. CHAPTER SUMMARY

Implementation of the fault tolerant sequencer in a physical hardware board was discussed in this chapter. A brief discussion on the complex process of moving from a Verilog or schematic design to an actual FPGA was discussed. Each step of this process, from synthesis to generating the final programing file for the FPGA, has dozens of internal steps. Each of these stages has many configuration options in software that control the optimization, speed, compatibility and format of the final product. Careful control of these settings is essential to avoid an "optimization" step removing all of the redundancy offered by these fault tolerant techniques.

A comparison of the key FPGA metrics was presented next, providing an overview of the actual hardware that is required by the sequencer. An example of the major drawback of fault tolerant designs, the cost in hardware resources, was provided in this section. Both fault tolerant designs consumed about three times the number of gates and FFs as the non-protected version. Any fault tolerant logic designer must keep this in mind, especially when using premade logic cores such as large processors or purpose-built cores. Triplicating these designs can quickly cause the engineer to run out of system resources.

Finally, a comparison was made of the various hardware boards available for testing at the time of writing. Given the various choices available, the Digilent Genesys board was the appropriate decision. This board provided all the necessary testing features to validate the operation of the sequencer and TMR features on physical hardware. This testing was performed and the design validated. This testing method demonstrated the overall operation of the sequencer but is somewhat limited since internal signals and operation cannot be verified once in hardware without adding a significant amount of additional signals and I/O devices to the testing hardware. The performance of the sequencer here meets the operational and design requirements specified in earlier chapters. A summary and recommendations for further development are found in the following chapter.

VI. CONCLUSION AND RECOMMENDATIONS FOR FUTURE WORK

The difficulties of operating complex electronic devices in a radiation environment like that encountered by spacecraft on orbit were discussed in this thesis. Radiation has some impact on all electronics, but the focus here was on FPGA technology. The design of a launch control sequencer suitable for use on the NPSCuL was detailed. A fault tolerant technique, TMR, was discussed and applied to the design using both manual methods and automatic software tools. The fault tolerances of the three final designs were compared. Implementation of the design in physical hardware was discussed and testing was performed on a development kit that verified the simulation results that showed the effectiveness of the manual TMR design in correcting SETs or SEUs.

A. CONCLUSIONS

New developments in FPGA manufacturing technology has produced radiation tolerant devices with significant resistance to upsets. Despite the high cost and complex features of these specialized FPGAs, these errors still occur. In order to combat these effects, various fault tolerant logic design methods were discussed. TMR, which has been proven to be a reliable fault tolerance method, was selected as the appropriate method for this design. TMR lends itself very well to a modular design, allowing its application to either small or large blocks of logic. While the design presented here is fairly small, the techniques scale with larger designs.

For this level of design, the manual TMR method is clearly superior in its ability to tolerate errors over the software and single TMR design. The very small increase in FPGA resources for the manual method is insignificant compared to the level of protection provided by replicating all of the major modules. The major FGPA manufactures are constantly improving their designs with more and more gates, and this serves to drive down the cost-per-gate for a given FPGA. The FPGAs considered for

similar applications should have more than sufficient overhead to support the most robust TMR design. This practice will also provide room for the future, allowing extra gates for new features and upgrades.

The sequencer developed here represents a specific application of the design techniques that should be employed for similar fault-tolerant projects. Separating the design into functional modules not only makes the development process easier, but it makes future changes or upgrades much easier. This approach also provides more opportunities for insertion of fault-tolerant features.

B. FOLLOW-ON RESEARCH

1. Radiation Testing

There are numerous areas that will require additional study and development before this sequencer design can be used in an actual mission. Additional testing and integration must be performed before the sequencer can be adapted to the existing NPSCuL hardware. In particular, the TMR version of the design requires actual radiation testing. While the most common or likely faults were simulated in this thesis, it cannot compare to testing in a real radiation environment. This task will require creation of a testing rig to allow communication while the FPGA is being irradiated and a method to detect and record the errors encountered. Retooling the existing design to support this very detailed monitoring will be a challenging task.

2. Place and Route Effects on MBU

As discussed in Chapter V, the effects of controlling the placement of logic blocks when they are being mapped onto an FPGA is an interesting direction of study. A carefully mapped placement of these blocks in a TMR type design is theoretically more resistant to radiation effects while not significantly affecting the performance of the design. This concept may be expanded beyond the humble sequencer design in the future.

3. Additional FPGA Features/Uses

One of the driving factors for using an FPGA in the sequencer design for the NPSCuL is the flexibility of the design. Not only can updates and improvements be performed quickly and without additional hardware testing, but new features can be added. Exploring some of the new features and capabilities possible with the NPSCuL after the CubeSats have been launched is an interesting area of study. Reusing the sequencer hardware once its primary mission is complete would be an excellent use of existing resources. The improved SADv3 design added several new memory options that are currently unused, and integration of these and other features presents an interesting opportunity for study.

4. Software Comparison

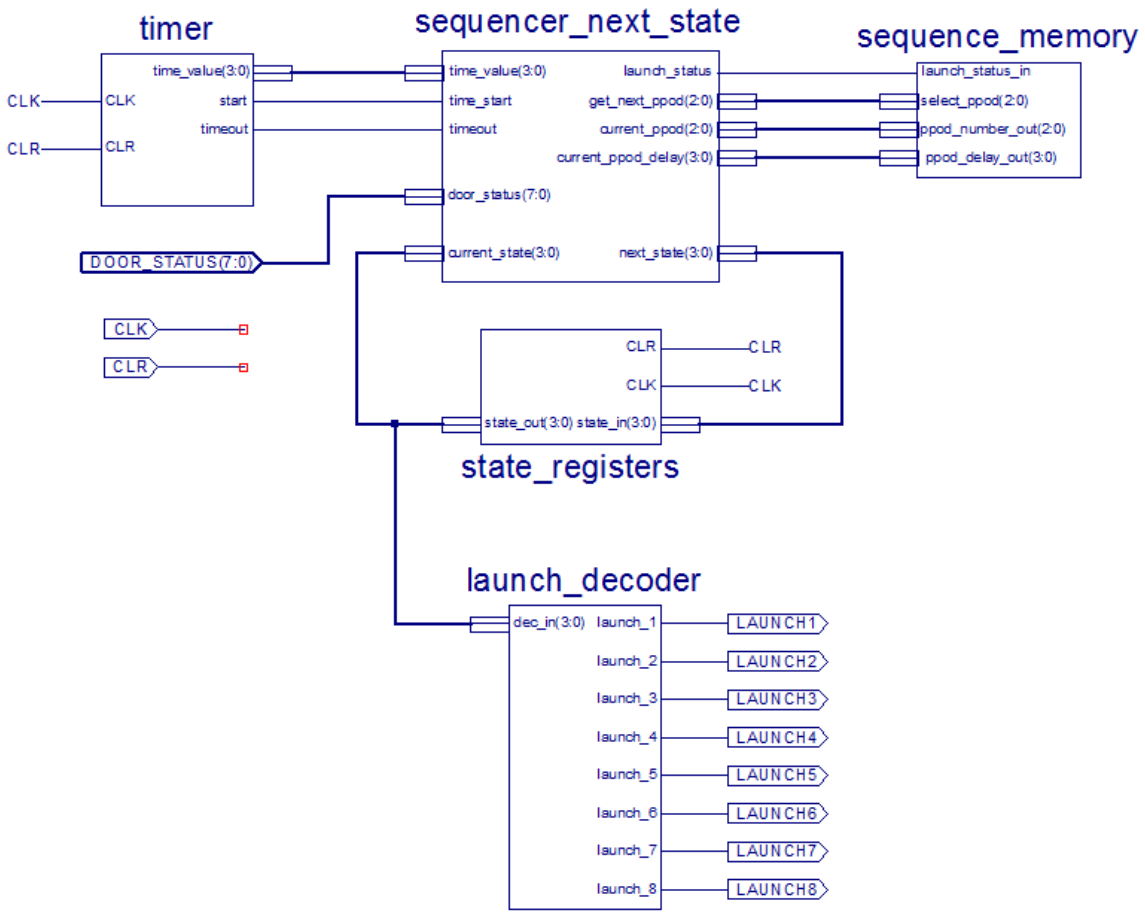
In this thesis, only one primary software package was used to perform the TMR functions. The three major FGPA manufacturers each offer TMR and fault-tolerance design features in their respective software packages. Due to ITAR restrictions, obtaining these software packages is difficult. A comparison of the features and performance of the various tools for this application would be useful for future designs.

THIS PAGE INTENTIONALLY LEFT BLANK

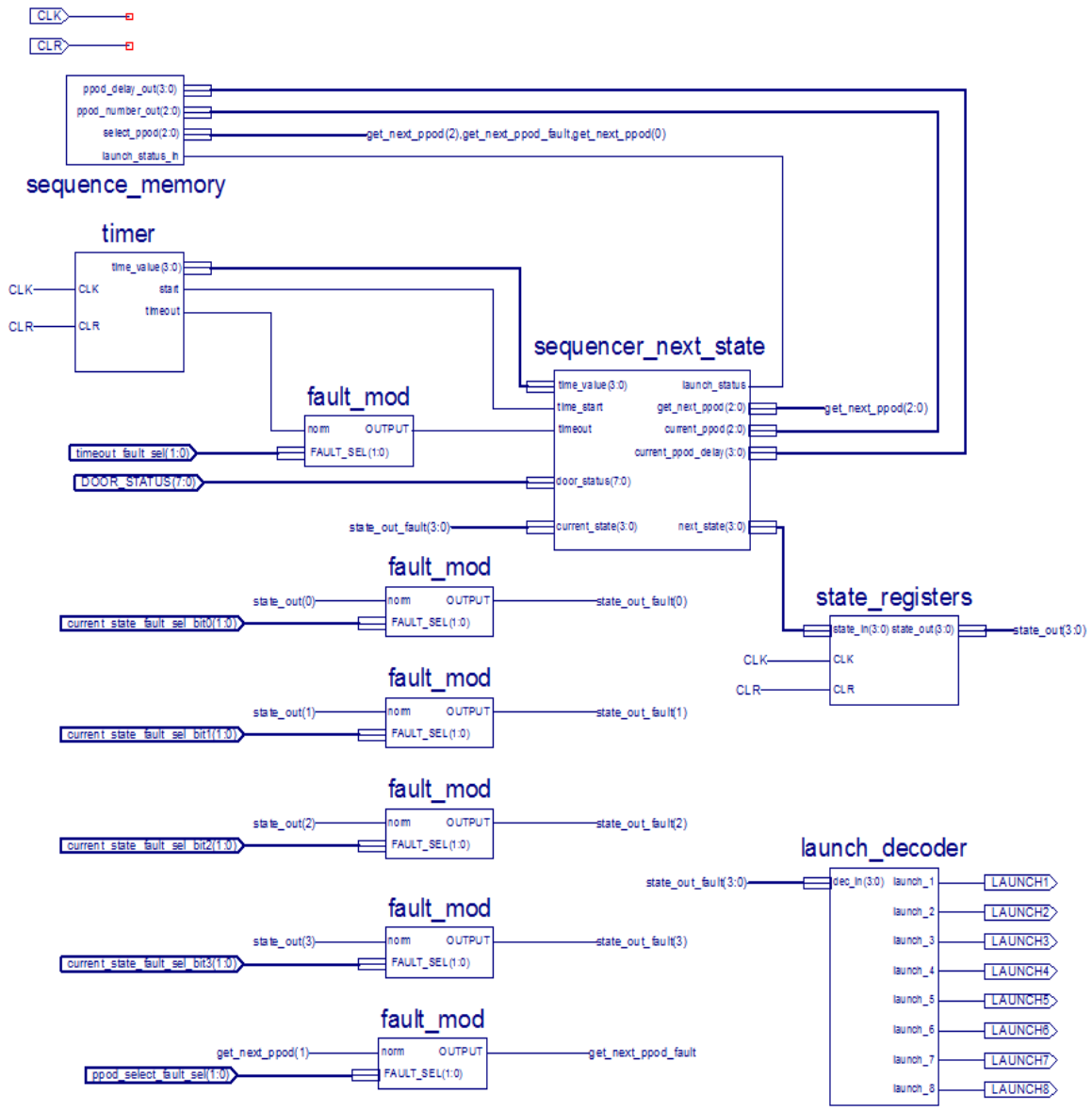
APPENDIX A. DESIGN SCHEMATICS

The sequencer design used a mix of Verilog code and Xilinx ISE schematic files. The schematic files for each of the three designs are contained in this appendix.

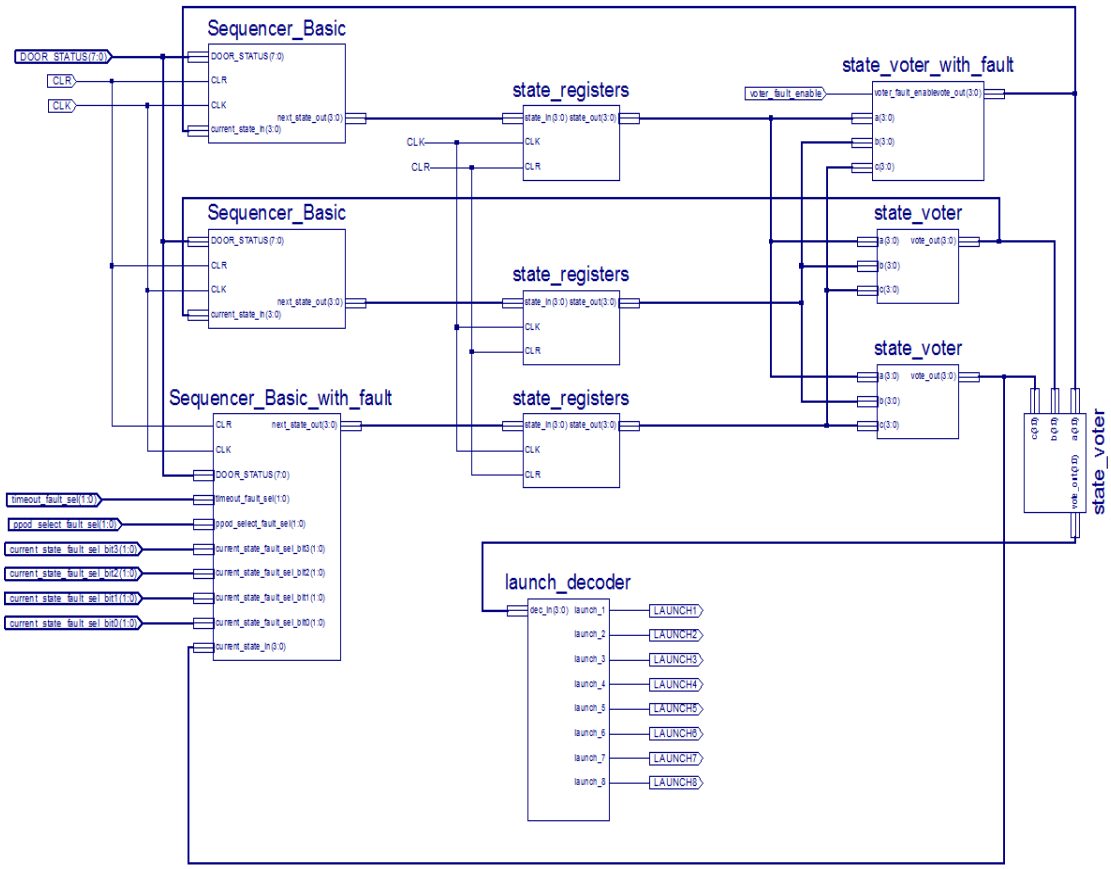
A. SINGLE SEQUENCER TOP LEVEL SCHEMATIC



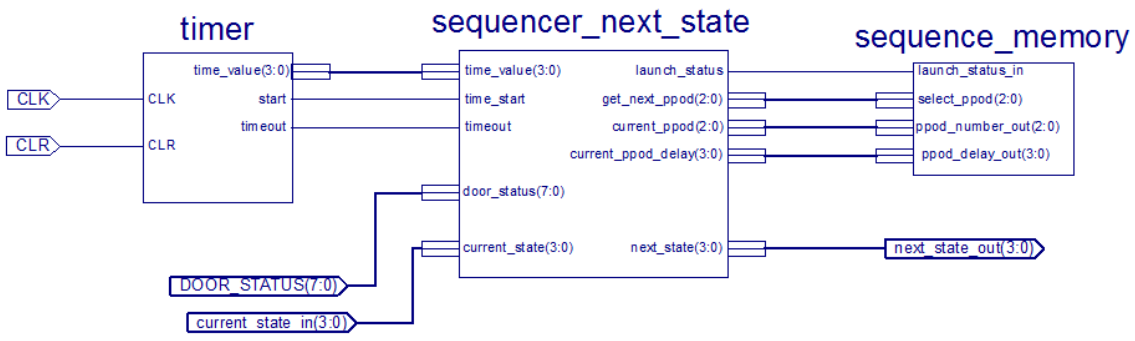
B. SINGLE SEQUENCER WITH FAULT MODULES TOP LEVEL SCHEMATIC



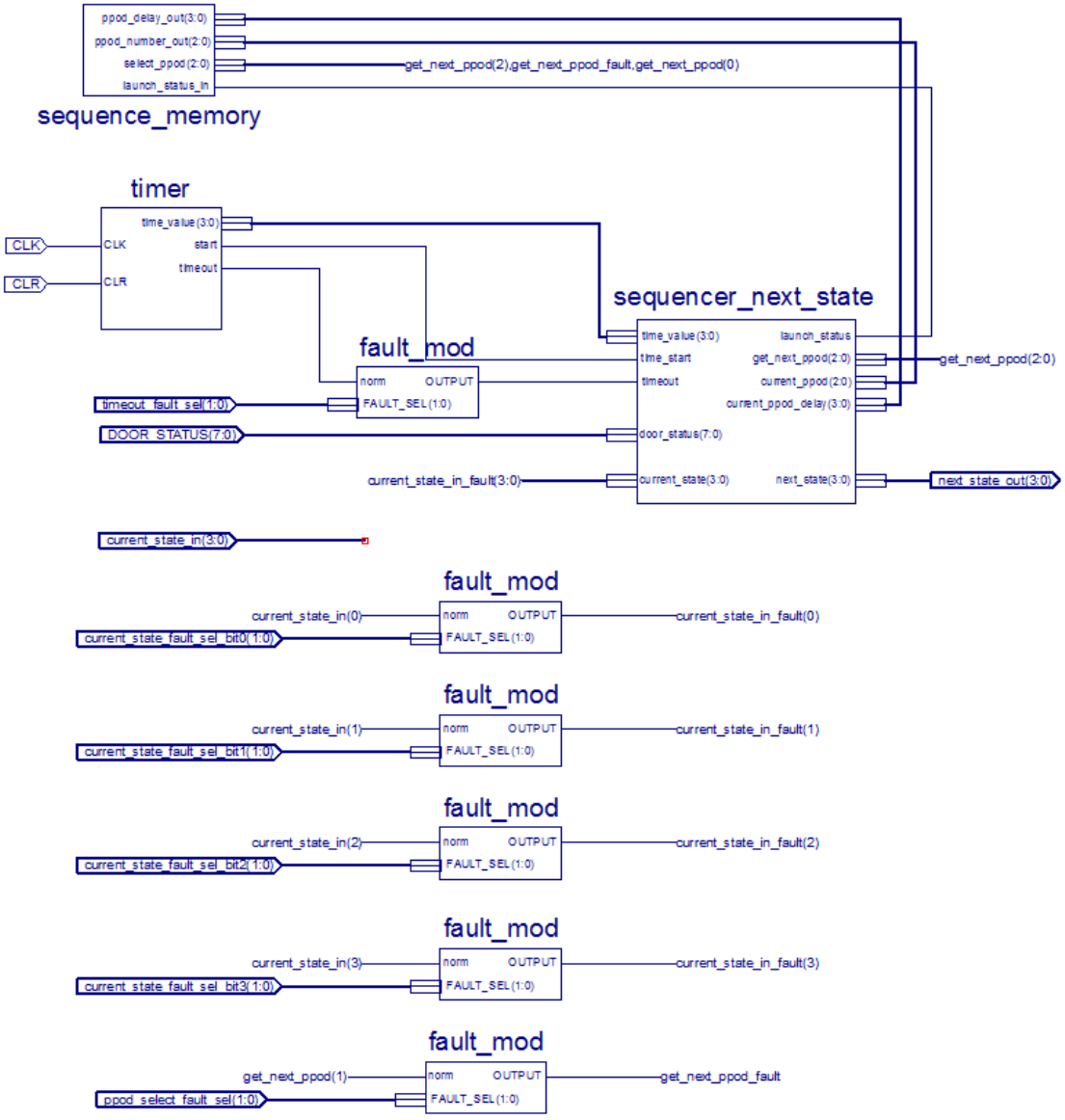
C. MANUAL TMR SEQUENCER WITH INTERNAL FAULT MODULES TOP LEVEL SCHEMATIC



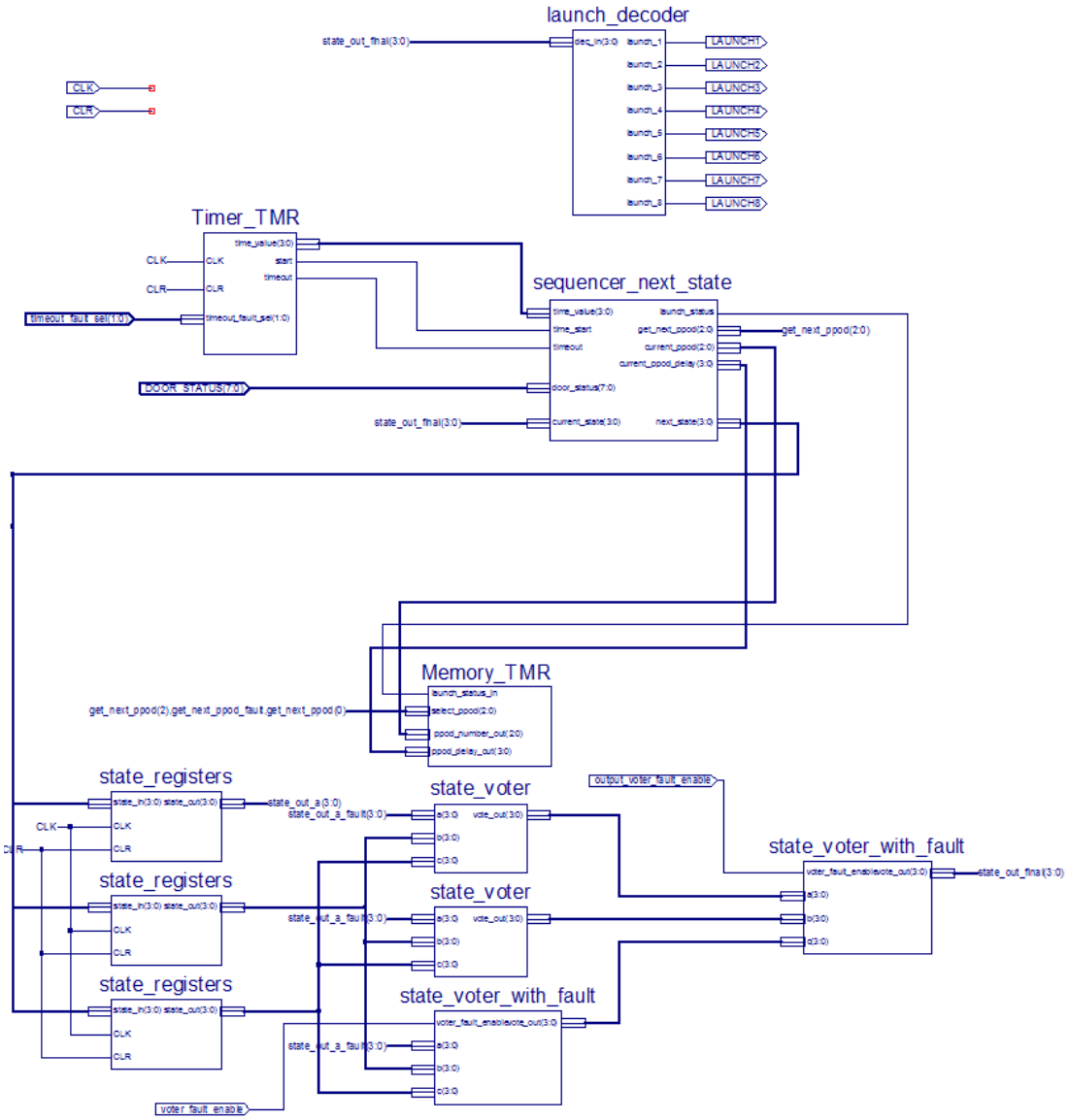
1. Manual TMR Basic Sequencer Module



2. Manual TMR Basic Sequencer Module with Fault Modules



D. SOFTWARE TMR SEQUENCER WITH FAULT MODULES



THIS PAGE LEFT INTENTIONALLY BLANK

APPENDIX B. VERILOG CODE

A. STATE REGISTERS

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////////
// Company: Naval Postgraduate School
// Engineer: LCDR Jason Brandt
//
// Create Date:    10:38:58 09/02/2013
// Design Name:    NPSCuL Sequencer
// Module Name:    state_registers
// Project Name:   Sequencer
// Target Devices:  Xilinx Virtex-5
// Tool versions:  ISE 14.6
// Description:  Four bit state registers with preset/clear
//
// Dependencies: None
//
// Revision: 2
// Revision 0.01 - File Created
// Additional Comments: None
//
//////////////////////////////////////////////////////////////////////////////////
////////////
module state_registers(
    input [3:0] state_in,
    output reg [3:0] state_out,
    input CLK,
    input CLR
    );

    always @(posedge CLK or posedge CLR) begin

        if (CLR) state_out = 8;  // clear sets START state

            else state_out = state_in;

        end

endmodule
```

B. TIMER MODULE

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////////
// Company: Naval Postgraduate School
// Engineer: LCDR Jason Brandt
//
// Create Date:    11:09:27 09/02/2013
// Design Name:    NPSCuL Sequencer
```

```
// Module Name:    timer
// Project Name:   Sequencer
// Target Devices: Xilinx Virtex-5
// Tool versions:  ISE 14.6
// Description:  Count-up timer with async clear/start, and timeout
signal
//
// Dependencies:
//
// Revision: 4
// Revision 0.01 - File Created
// Additional Comments: None
//
//////////////////////////////////////////////////////////////////////
////////////
module timer(
    input [3:0] time_value,
    input start,
    input CLK,
     input CLR,

     output reg timeout = 0
    );

    reg [3:0] counter = 0;
    reg [3:0] init_counter = 0;

    always @(posedge start or posedge CLR) begin
        counter = 0;
        if(CLR) init_counter = 0;
            else if(start)  init_counter = time_value;
                else    init_counter = 0;
        end

    always @(posedge CLK) begin

        if(timeout) timeout <= 0;

        else if(counter == init_counter)
                timeout <= 1;

                else begin
                    counter <= counter + 1;
                    timeout <= 0;
                    end

        end

endmodule
```

C. NEXT STATE MODULE

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////////
// Company: Naval Postgraduate School
// Engineer: LCDR Jason Brandt
//
// Create Date:    12:41:12 08/24/2013
// Design Name:    NPSCuL Sequencer
// Module Name:    Sequencer_Next_State
// Project Name:   Sequencer
// Target Devices: Xilinx Virtex-5
// Tool versions:  ISE 14.6
// Description:  Next-state logic for sequencer
//
// Dependencies: None
//
// Revision: 8
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////////////
module sequencer_next_state(

    // Door Status Switch Inputs from door hardware
     // Assume: 0 = Door SHUT, 1 = Door OPEN
     input [7:0] door_status,
     input [2:0] current_ppod, // PPod being launched
     input [3:0] current_ppod_delay, // delay for current PPod
     input timeout,  // input from timer

     input CLR,

     input [3:0] current_state,  // input from state register
     output reg [3:0] next_state, // output to state register

     output reg [2:0] get_next_ppod = 0, // get next ppod number and
delay from storage

     output reg launch_status, // store launch status in storage

     output reg [3:0] time_value, // outputs for timer
     output reg time_start
     );

    // Define States
    reg ppod_advance = 0;

    parameter INIT = 8;
    parameter WAIT = 9;
    parameter ADVANCE = 10;
    parameter LAUNCH_FAIL = 11;
    parameter LAUNCH_SUCCESS = 12;
```

```
    parameter DONE = 13;
    parameter LAUNCH1 = 0;
    parameter LAUNCH2 = 1;
    parameter LAUNCH3 = 2;
    parameter LAUNCH4 = 3;
    parameter LAUNCH5 = 4;
    parameter LAUNCH6 = 5;
    parameter LAUNCH7 = 6;
    parameter LAUNCH8 = 7;
    parameter START = 14;

    parameter ppod_door_delay = 5; // default wait for door to open

    always @ (*) begin
        if(CLR) begin
            ppod_advance <= 0;
            time_start <= 0;
            next_state <= START;
            get_next_ppod <= 0;

            end


        case (curresnt_state)

            default: begin
                next_state <= START;
                end

            INIT: begin  // additional wait state to allow timeout to
clear
                time_start <= 0;
                next_state <= START;
                end

            START: begin
                    time_value <= current_ppod_delay; // start launch
timer
                    time_start <= 1;
                    next_state <= WAIT;
                end

            WAIT: begin

                if(timeout) begin
                    time_value <= ppod_door_delay; // set timer for
ppod door timeout
                    time_start <= 0;
                    next_state <= current_ppod; // jump to launch state
based on PPOD #
                    end

                    else
                        next_state <= WAIT;
```

```
    end

ADVANCE: begin
        time_start <= 0;
        next_state <= START;
        if(ppod_advance) begin  // advance sequence
            if(get_next_ppod == 7) next_state <= DONE;
                else begin
                launch_status <= 0;
                get_next_ppod <= get_next_ppod + 1;
                ppod_advance <= 0;
                end
            end
    end

LAUNCH_FAIL: begin
    launch_status <= 0;
    ppod_advance <= 1;
    next_state <= ADVANCE;
    end

LAUNCH_SUCCESS: begin
    launch_status <= 1;
    ppod_advance <= 1;
    next_state <= ADVANCE;
    end

DONE: begin
    next_state <= DONE;
    end

LAUNCH1: begin
    time_start <= 1; // start door timer
    if(door_status[0])
        next_state <= LAUNCH_SUCCESS;

        else if(timeout)
            next_state <= LAUNCH_FAIL;

                else
                    next_state <= LAUNCH1;

    end

LAUNCH2: begin
    time_start <= 1; // start door timer
    if(door_status[1])
        next_state <= LAUNCH_SUCCESS;

        else if(timeout)
            next_state <= LAUNCH_FAIL;

                else
                    next_state <= LAUNCH2;
```

```
    end
LAUNCH3: begin
    time_start <= 1; // start door timer
    if(door_status[2])
        next_state <= LAUNCH_SUCCESS;

        else if(timeout)
            next_state <= LAUNCH_FAIL;

                else
                    next_state <= LAUNCH3;

    end
LAUNCH4: begin
    time_start <= 1; // start door timer
    if(door_status[3])
        next_state <= LAUNCH_SUCCESS;

        else if(timeout)
            next_state <= LAUNCH_FAIL;

                else
                    next_state <= LAUNCH4;

    end
LAUNCH5: begin
    time_start <= 1; // start door timer
    if(door_status[4])
        next_state <= LAUNCH_SUCCESS;

        else if(timeout)
            next_state <= LAUNCH_FAIL;

                else
                    next_state <= LAUNCH5;

    end
LAUNCH6: begin
    time_start <= 1; // start door timer
    if(door_status[5])
        next_state <= LAUNCH_SUCCESS;

        else if(timeout)
            next_state <= LAUNCH_FAIL;

                else
                    next_state <= LAUNCH6;

    end
LAUNCH7: begin
    time_start <= 1; // start door timer
    if(door_status[6])
        next_state <= LAUNCH_SUCCESS;

        else if(timeout)
```

```
                        next_state <= LAUNCH_FAIL;

                            else
                                next_state <= LAUNCH7;

                end
            LAUNCH8: begin
                time_start <= 1; // start door timer
                if(door_status[7])
                    next_state <= LAUNCH_SUCCESS;

                    else if(timeout)
                        next_state <= LAUNCH_FAIL;

                            else
                                next_state <= LAUNCH8;

                end

        endcase

    end

endmodule
```

D. SEQUENCE MEMORY MODULE

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////////
// Company: Naval Postgraduate School
// Engineer: LCDR Jason Brandt
//
// Create Date:    10:04:10 09/02/2013
// Design Name:    NPSCuL Sequencer
// Module Name:    Sequence_Memory
// Project Name:   Sequencer
// Target Devices: Xilinx Virtex-5
// Tool versions:  ISE 14.6
// Description:  Memory module sets launch sequence and delay times
//
// Dependencies: None
//
// Revision: 2
// Revision 0.01 - File Created
// Additional Comments: None
//
//////////////////////////////////////////////////////////////////////////////////
////////////
module sequence_memory(
    input [2:0] select_ppod,
     input launch_status_in,
    output reg [2:0] ppod_number_out,
    output reg [3:0] ppod_delay_out
    );
```

```
reg [7:0] launch_status = 0;

parameter LAUNCH_SEQ_1 = 0;  // set sequence here
parameter LAUNCH_SEQ_2 = 1;
parameter LAUNCH_SEQ_3 = 2;
parameter LAUNCH_SEQ_4 = 3;
parameter LAUNCH_SEQ_5 = 4;
parameter LAUNCH_SEQ_6 = 5;
parameter LAUNCH_SEQ_7 = 6;
parameter LAUNCH_SEQ_8 = 7;

parameter LAUNCH_DEL_1 = 5;  // set time delay here
parameter LAUNCH_DEL_2 = 10;
parameter LAUNCH_DEL_3 = 15;
parameter LAUNCH_DEL_4 = 5;
parameter LAUNCH_DEL_5 = 10;
parameter LAUNCH_DEL_6 = 15;
parameter LAUNCH_DEL_7 = 5;
parameter LAUNCH_DEL_8 = 10;


always @ (*) begin

    launch_status[ppod_number_out] <= launch_status_in;

    case (select_ppod)

        0: begin
               ppod_number_out = LAUNCH_SEQ_1;
               ppod_delay_out = LAUNCH_DEL_1;
           end

        1: begin
               ppod_number_out = LAUNCH_SEQ_2;
               ppod_delay_out = LAUNCH_DEL_2;
           end

        2: begin
               ppod_number_out = LAUNCH_SEQ_3;
               ppod_delay_out = LAUNCH_DEL_3;
           end

        3: begin
               ppod_number_out = LAUNCH_SEQ_4;
               ppod_delay_out = LAUNCH_DEL_4;
           end

        4: begin
               ppod_number_out = LAUNCH_SEQ_5;
               ppod_delay_out = LAUNCH_DEL_5;
           end

        5: begin
               ppod_number_out = LAUNCH_SEQ_6;
```

```
                ppod_delay_out = LAUNCH_DEL_6;
            end

        6: begin
                ppod_number_out = LAUNCH_SEQ_7;
                ppod_delay_out = LAUNCH_DEL_7;
            end

        7: begin
                ppod_number_out = LAUNCH_SEQ_8;
                ppod_delay_out = LAUNCH_DEL_8;
            end

    endcase

end

endmodule
```

E. LAUNCH DECODER

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////////
// Company: Naval Postgraduate School
// Engineer: LCDR Jason Brandt
//
// Create Date:    14:26:47 09/08/2013
// Design Name:    NPSCuL Sequencer
// Module Name:    launch_decoder
// Project Name:   Sequencer
// Target Devices: Xilinx Virtex-5
// Tool versions:  ISE 14.6
// Description:  Three-bit decoder
//
// Dependencies: None
//
// Revision: 1
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////////////
module launch_decoder(
    input [3:0] dec_in,
    output launch_1,
     output launch_2,
     output launch_3,
     output launch_4,
     output launch_5,
     output launch_6,
     output launch_7,
     output launch_8
    );
```

```
    and(launch_1,~dec_in[3],~dec_in[2],~dec_in[1],~dec_in[0]);
    and(launch_2,~dec_in[3],~dec_in[2],~dec_in[1],dec_in[0]);
    and(launch_3,~dec_in[3],~dec_in[2],dec_in[1],~dec_in[0]);
    and(launch_4,~dec_in[3],~dec_in[2],dec_in[1],dec_in[0]);
    and(launch_5,~dec_in[3],dec_in[2],~dec_in[1],~dec_in[0]);
    and(launch_6,~dec_in[3],dec_in[2],~dec_in[1],dec_in[0]);
    and(launch_7,~dec_in[3],dec_in[2],dec_in[1],~dec_in[0]);
    and(launch_8,~dec_in[3],dec_in[2],dec_in[1],dec_in[0]);

endmodule
```

F. ONE-BIT VOTER MODULE

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////////
// Company: Naval Postgraduate School
// Engineer: LCDR Jason Brandt
//
// Create Date:    13:43:50 09/08/2013
// Design Name:    NPSCuL Sequencer
// Module Name:    voter_one_bit
// Project Name:   Sequencer
// Target Devices: Xilinx Virtex-5
// Tool versions:  ISE 14.6
// Description: One bit majority voter
//
// Dependencies: None
//
// Revision: 1
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////////////
module voter_one_bit(
    input a,
    input b,
    input c,
    output vote_out
    );

    wire a_1;
    wire b_1;
    wire c_1;

    and(a_1, a, b);
    and(b_1, a, c);
    and(c_1, b, c);

    or(vote_out, a_1, b_1, c_1);


endmodule
```

G. STATE VOTER MODULE

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////////
// Company: Naval Postgraduate School
// Engineer: LCDR Jason Brandt
//
// Create Date:    13:40:09 09/08/2013
// Design Name:    NPSCuL Sequencer
// Module Name:    state_voter
// Project Name:   Sequencer
// Target Devices: Xilinx Virtex-5
// Tool versions:  ISE 14.6
// Description:  Four bit majority voter
//
// Dependencies: voter_one_bit
//
// Revision: 1
// Revision 0.01 - File Created
// Additional Comments: None
//
//////////////////////////////////////////////////////////////////////////////////
////////////
module state_voter(
    input [3:0] a,
    input [3:0] b,
    input [3:0] c,
    output [3:0] vote_out
    );
    voter_one_bit bit0 (
        .a(a[0]),
        .b(b[0]),
        .c(c[0]),
        .vote_out(vote_out[0])
        );
    voter_one_bit bit1 (
        .a(a[1]),
        .b(b[1]),
        .c(c[1]),
        .vote_out(vote_out[1])
        );
    voter_one_bit bit2 (
        .a(a[2]),
        .b(b[2]),
        .c(c[2]),
        .vote_out(vote_out[2])
        );
    voter_one_bit bit3 (
        .a(a[3]),
        .b(b[3]),
        .c(c[3]),
        .vote_out(vote_out[3])
        );
endmodule
```

H. STATE VOTER WITH FAULT INSERTION

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////////
// Company: Naval Postgraduate School
// Engineer: LCDR Jason Brandt
//
// Create Date:    13:23:09 10/08/2013
// Design Name:    NPSCuL Sequencer
// Module Name:    state_voter_with_fault
// Project Name:   Sequencer
// Target Devices: Xilinx Virtex-5
// Tool versions:  ISE 14.6
// Description:  Four bit majority voter with insertable fault
//
// Dependencies: voter_one_bit
//
// Revision: 2
// Revision 0.01 - File Created
// Additional Comments: None
//
//////////////////////////////////////////////////////////////////////////////////
////////////

module state_voter_with_fault(
    input [3:0] a,
    input [3:0] b,
    input [3:0] c,
     input voter_fault_enable,
    output [3:0] vote_out
    );

    wire bit0_internal;

    voter_one_bit bit0 (
        .a(a[0]),
        .b(b[0]),
        .c(c[0]),
        .vote_out(bit0_internal)
        );

    voter_one_bit bit1 (
        .a(a[1]),
        .b(b[1]),
        .c(c[1]),
        .vote_out(vote_out[1])
        );
    voter_one_bit bit2 (
        .a(a[2]),
        .b(b[2]),
        .c(c[2]),
        .vote_out(vote_out[2])
        );
    voter_one_bit bit3 (
```

```
        .a(a[3]),
        .b(b[3]),
        .c(c[3]),
        .vote_out(vote_out[3])
        );

    assign vote_out[0] = (voter_fault_enable) ? ~bit0_internal :
bit0_internal;

endmodule
```

I. BEHAVIORAL TEST FIXTURE

```
// Verilog test fixture created from schematic C:\Users\Jason
Brandt\Dropbox\NPS\Thesis\Single_Sequencer_Test_Rev3\Sequencer_Basic.sc
h - Sat Nov 02 06:56:44 2013

`timescale 1ms / 1ms

module Sequencer_Basic_Sequencer_Basic_sch_tb();

// Inputs
   reg CLR;
   reg CLK;
   reg [7:0] DOOR_STATUS;
   reg [1:0] timeout_fault_sel;
   reg [1:0] ppod_select_fault_sel;
   reg [1:0] current_state_fault_sel_bit2;
   reg [1:0] current_state_fault_sel_bit3;
   reg [1:0] current_state_fault_sel_bit1;
   reg [1:0] current_state_fault_sel_bit0;

// Output
   wire LAUNCH1;
   wire LAUNCH2;
   wire LAUNCH3;
   wire LAUNCH4;
   wire LAUNCH5;
   wire LAUNCH6;
   wire LAUNCH7;
   wire LAUNCH8;

// Bidirs

// Instantiate the UUT
   Sequencer_Basic UUT (
        .CLR(CLR),
        .CLK(CLK),
        .DOOR_STATUS(DOOR_STATUS),
        .timeout_fault_sel(timeout_fault_sel),
        .ppod_select_fault_sel(ppod_select_fault_sel),
        .current_state_fault_sel_bit2(current_state_fault_sel_bit2),
        .current_state_fault_sel_bit3(current_state_fault_sel_bit3),
        .current_state_fault_sel_bit1(current_state_fault_sel_bit1),
        .current_state_fault_sel_bit0(current_state_fault_sel_bit0),
```

```
        .LAUNCH1(LAUNCH1),
        .LAUNCH2(LAUNCH2),
        .LAUNCH3(LAUNCH3),
        .LAUNCH4(LAUNCH4),
        .LAUNCH5(LAUNCH5),
        .LAUNCH6(LAUNCH6),
        .LAUNCH7(LAUNCH7),
        .LAUNCH8(LAUNCH8)
    );
// Initialize Inputs

       initial begin
        CLR = 0;
        CLK = 0;
        DOOR_STATUS = 0;
        timeout_fault_sel = 0;
        ppod_select_fault_sel = 0;
        current_state_fault_sel_bit3 = 0;
        current_state_fault_sel_bit2 = 0;
        current_state_fault_sel_bit1 = 0;
        current_state_fault_sel_bit0 = 0;



        #2
        CLR = 1;
        DOOR_STATUS = 255;
        #4
        CLR = 0;
        // Uncomment relevant section to enable specific faults

//    Timeout fault
//      #200
//      timeout_fault_sel = 1;
//      #60
//      timeout_fault_sel = 0;

//      PPOD select fault
//      #200
//      ppod_select_fault_sel = 1;
//      #60
//      ppod_select_fault_sel = 0;

//      State Output fault - MSB Case #1
//      #200
//      current_state_fault_sel_bit3 = 1;
//      #60
//      current_state_fault_sel_bit3 = 0;

//      State Output fault - MSB Case #2
//      #140
//      current_state_fault_sel_bit3 = 1;
//      #10
//      current_state_fault_sel_bit3 = 0;
```

```
//      State Output fault - LSB Case #1
//      #140
//      current_state_fault_sel_bit0 = 1;
//      #60
//      current_state_fault_sel_bit0 = 0;

        end

    always begin
    #1
    CLK = ~CLK;
    end

endmodule
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] A. D. Harris, "NPS Cubesat launcher-lite sequencer," M.S. thesis, Naval Postgraduate School, Monterey, CA, 2009.

[2] L. S. Parobek, "Research, development & testing of a fault-tolerant FPGA based sequencer for CubeSAT launching applications," M.S. thesis, Naval Postgraduate School, Monterey, CA, 2013.

[3] D. A. Ebert, "Design and development of a configurable fault-tolerant processor (CFTP) for space applications," M.S. thesis, Naval Postgraduate School, Monterey, CA, 2003.

[4] R. Yuan, "Triple modular redundancy (TMR) in a configurable fault-tolerant processor (CFTP) for space applications," M. S. thesis, Naval Postgraduate School, Monterey, CA, 2003.

[5] J. D. Snodgrass, "Low-power fault tolerance for spacecraft FPGA-based numerical computing," PhD dissertation, Naval Postgraduate School, Monterey, CA, 2006.

[6] M. A. Sullivan, "Employment of reduced precision redundancy for fault tolerant FPGA applications," IEEE Symposium on Field Programmable Custom Computing Machines, Monterey, CA, 2009.

[7] P. J. Majewicz, "Implementation of a Configurable Fault Tolerant Processor (CFTP) using Internal Triple Modular Redundancy (TMR)," M.S. thesis, Naval Postgraduate School, Monterey, CA, 2005.

[8] National Areonautics and Space Administration, "NASA CubeSat launch initiative." [Online]. Available: http://www.nasa.gov/directorates/heo/home/CubeSats_initiative.html. [Accessed 31 October 2013].

[9] The CubeSat Program, "CubeSat design specification, Revision 12," California Polytechnic State University, San Luis Obispo, CA, 2009.

[10] A. DeJesus, "NPS CubeSat Launcher Program Update," in *CubeSat Developers Workshop*, Monterey, CA, 2009.

[11] National Aeronautical Space Administration. (February 2013). "NASA's Van Allen Probes Reveal Previously Undetected Radiation Belt Around Earth," [Online]. Available: http://www.nasa.gov/mission_pages/sunearth/news/gallery/20130228-radiationbelts.html#.UmBZofnbNCg. [Accessed 02 October 2013].

[12] B. Bridgford, C. Carmichael and C. W. Tseng, “Single Event Upset Mitigation Selection Guide” (XAPP 987) v1.0, *Xilinx Application Notes*, Mar. 2008.

[13] J. E. Parks, “The Compton Effect—Compton Scattering and Gamma Ray Spectroscopy,” Department of Physics, University of Tennessee, Knoxville, TN, 2009.

[14] F. B. Mclean and T. R. Oldham, “Basic Mechanisms of Radiation Effects in Electronic Materials and Devices,” Harry Diamond Laboratories, Adelphi, MD, 1987.

[15] J. R. Schwank, M. R. Shaneyfelt, D. M. Fleetwood, J. A. Felix, P. E. Dodd, P. Paillet and V. Ferlet-Cavrois, “Radiation Effects in MOS Oxides,” *IEEE Transactions of Nuclear Science*, vol. 55, no. 4, 2008.

[16] D. White, “Considerations Surrounding Single Event Effects in FPGAs, ASICs and Processors,” Xilinx, San Jose, CA, White Paper (WP402), 2012.

[17] E. P. Wilcox, S. D. Phillips, P. Cheng, T. Thrivikraman, A. Madan, J. D. Cressler, G. Vizkelethy, P. W. Marshall, C. Marshall, J. A. Babcock, K. Kruckmeyer, R. Eddy, G. Cestra and B. Zhang, “Single Event Transient Hardness of a New Complementary (npn + pnp) SiGe HBT Technology on Thick-Film SOI,” *IEEE Transactions on Nuclear Science,* vol. 57, no. 6, 2010.

[18] J. R. Schwank, V. Ferlet-Cavrois, M. R. Shanyfelt, P. Paillet and P. E. Dodd, “Radiation Effects in SOI Technologies,” *IEEE Transactions on Nuclear Science*, vol. 50, no. 3, 2003.

[19] K. Haque, “Radiation Hard FPGA Configuration Techniques using Silicon on Sapphire,” M.S. thesis, RMIT University, Melbourne, Australia, 2011.

[20] “UltraCMOS® Process Technology Overview,” Peregrine Semiconductor. [Online]. Available: http://www.psemi.com/content/ultracmos-process/ultracmos-process-tech.php. [Accessed 18 October 2013].

[21] D. Nenni, “The Semiconductor Wiki Project.” (26 August 2012). [Online]. Available: http://www.semiwiki.com/forum/content/1596-brief-history-field-programmable-devices-fpgas.html. [Accessed 19 October 2013].

[22] Xilinx, *Xilinx 7 Series FPGAs Overview Datasheet*, Xilinx, 2013.

[23] K. Underwood, “FPGAs vs CPUs: Trends in Peak Floating Point Performance,” Sandia National Laboratories, Albuquerque, NM, 2004.

[24] Altium, *Altium Designer’s Guide—FPGA Design Basics*, 2008.

[25] J. J. Wang, “Radiation Effects in FPGAs,” Actel Corporation, Mountain View, CA, 2008.

[26] J. Frank Hall Schmidt, “Fault Tolerant Design Implementation on Radiation Hardened By Design SRAM-Based FPGAs,” M.S. thesis, Massachusetts Institute of Technology, Boston, MA, 2013.

[27] T. Miyahira and G. Swift, “Evaluation of Radiation Effects in Flash Memories,” in *MAPLD Conference*, Greenbelt, MD, 1998.

[28] N. H. Rollins, “Hardware and Software Fault-Tolerance of Softcore Processors Implemented in SRAM-Based FPGAs,” PhD dissertaion, Brigham Young University, Provo, UT, 2012.

[29] Xilinx, *Radiation-Hardened, Space-Grade Virtex-5QV Family Overview Datasheet*, Xilinx, San Jose, CA, 2012.

[30] M. Niknahad, O. Sander and J. Becker, “QDFR—An Integration of Quadded Logic for Modern FPGAs to Tolerate High Radiation Effect Rates,” in *Radiation and Its Effects on Components and Systems (RADECS), 12th European Conference*, Sevilla, Spain, 2011.

[31] J. Tryon, “Redundant Logic Circuitry.” U.S. Patent 2,942,193, 30 July 1958.

[32] P. A. Jensen, “Quadded NOR Logic,” *IEEE Transtions on Reliability*, vols. R-12, no. 3, pp. 22–31, 1963.

[33] J. Von Neumann, “Probabilistic Logics and the synthesis of relaible organism from unreliable components,” in *Automata Studies*, no 34. Princeton, NJ: Princeton University Press, 1956.

[34] R. E. Lyons and W. Vanderkulk, “The Use of Triple-Modular Redundancy to Improve Computer Reliability,” *IBM Journal of Research and Development*, vol. 6, no. 2, 1962.

[35] M. Niknahad, O. Sander and J. Becker, “Fine Grain Fault Tolerance—A Key to High Reliability for FPGAs in Space,” in *Aerospace Conference, 2012 IEEE*, Big Sky, MT, 2012.

[36] J. Han and P. Jonker, “A Study On Fault-Tolerant Circuits Using Redundancy,” Delft University of Technology, Delft, The Netherlands.

[37] B. Sklar and P. K. Ray, *Digital Communications: Fundamentals and Applications*. Upper Saddle River, NJ: Prentice Hall, 2001.

[38] B. Shim, S. Sridhara and N. Shanbhag, “Reliable low-power digitial signal processing via reduced precision redundancy,” *IEEE Transactions on Very Large Scale Integration Systems*, vol. 12, no. 5, pp. 497–500, 2004.

[39] B. Pratt, “SEU Mitigation,” Brigham Young University. (2012). [Online]. Available: http://www.brianhpratt.net/cms/index.php?page=seu-mitigation. [Accessed 19 October 2013].

[40] J. Emmert, C. Skaggs and M. Abramovici, “Dynamic Fault Tolerance in FPGAs via Partial Reconfiguration,” in *IEEE Symposium on Custom Computing Machines*, Napa, CA, 2000.

[41] C. M. Hicks, “NPS Cubesat Launcher Program Management,” M.S. thesis, Naval Postgraduate School, Monterey, CA, 2009.

[42] G. Burdis, “The Atlas V Aft Bulkhead Carrier - Requirements for the Small Satellite Designer,” in *24th Annual AIAA/USU Conference on Small Satellites*, Logan, UT, 2010.

[43] T. C. Program, *Poly Picosatellite Orbital Deployer Mk III ICD*. San Luis Obispo, CA: California Polytechnic State University, 2007.

[44] F. Irom, F. Farmanesh, M. White and C. K. Kouba, *Frequency Dependence of Single-Event Upset in Highly Advanced PowerPC Microprocessors.* Pasadena, CA: California Institute of Technology, NASA JPL, 2006.

[45] Microsemi Corporation, “MHS Series 5 amp solid state relay datasheet,” Microsemi Corporation, Aliso Viejo, CA, 2012.

[46] C. Carmichael, E. Fuller, P. Blain and M. Caffrey, “SEU Mitigation Techniques for Virtex FPGAs in Space Applications,” Xilinx, Los Alamos, NM, 1991.

[47] D. J. Smith, “VHDL & Verilog Compared & Contrasted.” (2003). [Online]. Available: http://www.angelfire.com/in/rajesh52/verilogvhdl.html. [Accessed 4 November 2013].

[48] P. P. Shirvani and E. J. McCluskey, “SEU characterization of digital circuits using weighted test programs,” Center for Reliable Computing, Stanford, CA, 2001.

[49] Microsemi Corporation, *Using Synplify to Design in Microsemi Radiation-Hardened FPGAs* (Application Note AC139). Aliso Viejo, CA: Microsemi Corporation, 2012.

[50] Microsemi, *Maximizing logic Utilization in eX, SX, and SX-A FPGA Devices Using cc Macros* (Application Note AC201). Aliso Viejo, CA: Microsemi Corporation, 2012.

[51] P. Adell and G. Allen, “Assessing and mitigating radiation effects in Xilinx FPGAs,” Jet Propulsion Laboratory, Pasadena, CA, 2008.

[52] Actel, *ARM Cortex M1 Enabled ProASIC3L Development Kit Datasheet*. Aliso Viejo, CA: Actel Corporation, 2009.

[53] Digilent, Inc., *Genesys Board Reference Manual*. Pullman, WA: Digilent, Inc., 2013.

[54] K. O’Niell, “Antifuse FPGA Technology: Best Option for Satellite Applications,” *COTS Journal,* 2003.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California